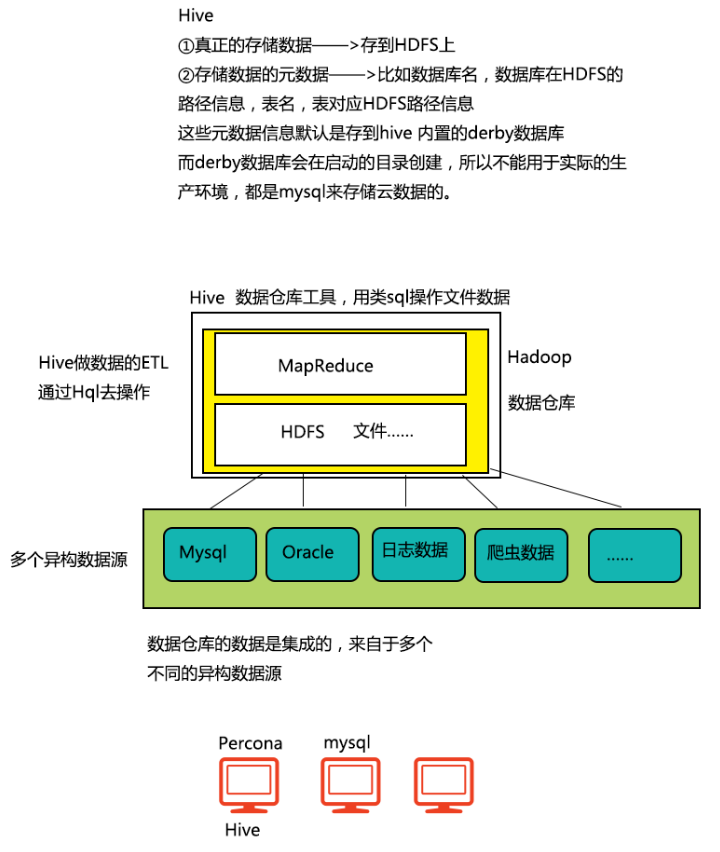
big\_day14

Hive01

**Hive介绍**



**Hadoop开发存在的问题**

只能用java语言开发，如果是c语言或其他语言的程序员用Hadoop，存在语言门槛。需要对Hadoop底层原理，api比较了解才能做开发。

**Hive概述**

Hive是基于Hadoop的一个**数据仓库工具**。可以将结构化的数据文件映射为一张表，并提供完整的sql查询功能，可以将sql语句转换为MapReduce任务进行运行。其优点是学习成本低，可以通过**类SQL**语句快速实现MapReduce统计，不必开发专门的MapReduce应用，十分适合数据仓库的统计分析。

Hive是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行**数据提取、转化、加载（ETL Extract-Transform-Load ）**，这是一种可以存储、查询和分析存在 Hadoop 中的大规模数据的机制。Hive 定义了简单的类 SQL 查询语言，称为 HiveQL，它允许熟悉 SQL 的用户查询数据。

**Hive的Hql**

HQL - Hive通过类SQL的语法，来进行分布式的计算。HQL用起来和SQL非常的类似，**Hive在执行的过程中会将HQL转换为MapReduce去执行**，所以Hive其实是基于Hadoop的一种分布式计算框架，**底层仍然是MapReduce**，所以它本质上还是一种离线大数据分析工具。

**数据仓库的特征**

1.数据仓库是多个异构数据源所集成的。

2.数据仓库存储的一般是**历史数据**。

3.数据库是为**捕获数据**而设计，数据仓库是为**分析数据**而设计。

4.数据仓库是时变的，数据存储从历史的角度提供信息。即数据仓库中的关键结构都隐式或显示地包含时间元素。

5.数据仓库是弱事务的，因为数据仓库存的是历史数据，一般都读（分析）数据场景。

**oltp和olap的比较**

数据库属于OLTP系统。（Online **Transaction** Processing）联机**事务**处理系统。涵盖了企业大部分的日常操作，如购物、库存、制造、银行、工资、注册、记账等。

数据仓库属于OLAP系统。（Online**Analytical** Processing）联机**分析**处理系统。

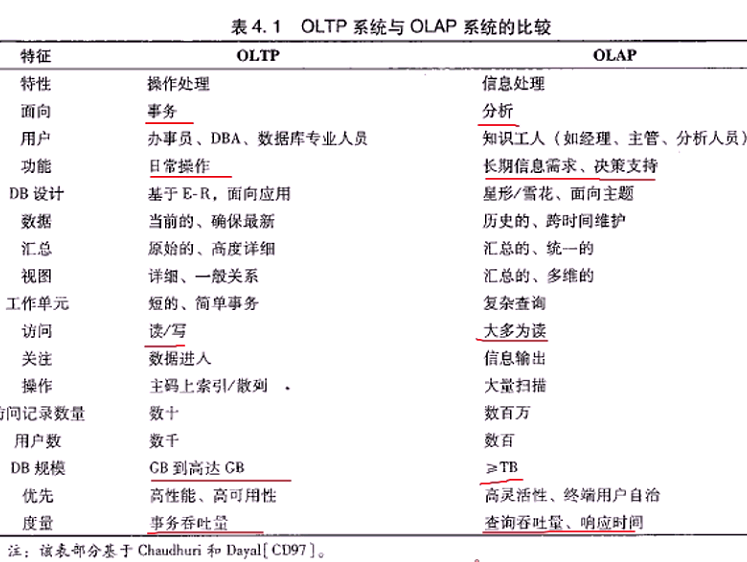
OLTP是面向用户的、用于程序员的事务处理以及客户的查询处理。

OLAP是面向市场的，用于知识工人（经理、主管和数据分析人员）的数据分析。

OLAP通常会集成多个异构数据源的数据，数量巨大。

OLTP系统的访问由于要保证原子性，所以有事务机制和恢复机制。

OLAP系统一般存储的是历史数据，所以大部分都是只读操作，不需要事务。



**适用场景**

Hive 构建在基于静态批处理的Hadoop 之上，Hadoop 通常都有较高的延迟并且在作业提交和调度的时候需要大量的开销。**因此，Hive 并不能够在大规模数据集上实现低延迟快速的查询**，例如，Hive 在几百MB 的数据集上执行查询一般有分钟级的时间延迟。因此，Hive 并不适合那些需要低延迟的应用，例如，联机事务处理(OLTP)。Hive 查询操作过程严格遵守Hadoop MapReduce 的作业执行模型，Hive 将用户的HiveQL 语句通过解释器转换为MapReduce 作业提交到Hadoop 集群上，Hadoop 监控作业执行过程，然后返回作业执行结果给用户。Hive 并非为联机事务处理而设计，Hive 并不提供实时的查询和基于行级的数据更新操作。**Hive 的最佳使用场合是大数据集的批处理作业，例如，网络日志分析**。

**Hive的安装**

基于JDK和Hadoop的环境变量

进入到bin目录，指定：sh hive  （或者执行：./hive）

IMG_258

**hive的基础指令**

|  |  |  |
| --- | --- | --- |
| 命令 | 作用 | 额外说明 |
| show  databases; | 查看都有哪些数据库 |  |
| create database park; | 创建park数据库 | 创建的数据库，实际是在Hadoop的HDFS文件系统里创建一个目录节点，统一存在： /user/hive/warehouse 目录下 |
| use  park; | 进入park数据库 |  |
| show tables; | 查看当前数据库下所有表 |  |
| create table stu   (id int,name string); | 创建stu表，以及相关的两个字段 | hive里，用的是string，不用char和varchar。此外，所创建的表，也是HDFS里的一个目录节点 |
| insert into stu values(1,'zhang') | 向stu表插入数据 | 1.HDFS不支持数据的修改和删除，但是在2.0版本后支持了数据追加。实际上，insert into 语句执行的是追加操作。  2.所以注意：hive支持查询，行级别的插入。不支持行级别的删除和修改。  3.hive的操作实际是执行一个job任务，调用的是Hadoop的MR。    4.插入完数据之后，发现HDFS stu目录节点下，多了一个文件,文件里存了插入的数据，所以得出结论，hive存储的数据，是通过HDFS的文件来存储的。 |
| select \* from stu | 查看表数据 | 也可以根据字段来查询，比如select  id from stu |
| drop table stu | 删除表 |  |
| select  \* from stu | 查询stu表数据 | 有时会出现这样的情况：  原因是：我们从外部导入的文件的数据格式是：  1 jary  2 rose  即第一列和第二列是以空格为分隔符的。  但是把数据导入到hive之后，hive并不知道分隔符是什么，所以就不能正确的切分数据。所以显示null。  解决办法：在hive创建表的时候，要指定分割符，并且这个分割符要和外部文件里的分割符一致。详见指令8 |
| load data local inpath '/home/software/1.txt' into table stu; | 通过加载文件数据到指定的表里 | 在执行完这个指令之后，发现hdfs stu目录下多了一个1.txt文件。由此可见，hive的工作原理实际上就是在管理hdfs上的文件，把文件里数据抽象成二维表结构，然后提供hql语句供程序员查询文件数据。  可以做这样的实验：不通过load 指令，而通过插件向stu目录下再上传一个文件，看下hive是否能将数据管理到stu表里。 |
| create table stu1(id int,name string) row format delimited fields terminated by '  '; | 创建stu1表，并指定分割符 空格。 | 此时，把外部数据导入hive，就可以正确查出数据了。 |
| desc  stu | 查看 stu表结构 |  |
| create table stu2 like stu | 创建一张stu2表，表结构和stu表结构相同 | like只复制表结构，不复制数据 |
| **insert  overwrite**  table stu2  select \* from stu | 把stu表数据插入到stu2表中 | insert  overwrite 可用于将select 查询出的数据插入到指定的表中或指定的目录下  比如：把查询结果存到本地指定的目录下，  执行：insert overwrite local directory '/home/stu' row format delimited fields terminated by  '  ' select \* from stu;  也可以将查询结果存到HDFS文件系统上  执行：insert overwrite directory '/stu' row format delimited fields terminated by  '  '  select \* from stu;  也可以将查询结果插入到多张表中  执行：from stu insert overwrite table stu1 select \* insert overwrite table stu2 select \*;  结果是把stu表的数据插入 stu1和stu2 表。（也可以加where 条件等，比如select \* where id>3） |
| alter table  stu rename to stu2 | 为表stu重命名为stu2 |  |
| alter table stu add columns (age int); | 为表stu增加一个列字段age，类型为int |  |
| exit | 退出hive | 当退出hive后，我们可以尝试做这样的一件事：  之前我们是在bin目录执行：sh hive 进入的  现在，我们换一个目录来进入hive,比如： sh /bin/hive 来进入  当我们查看数据库或查看表时，发现之前建立的park和stu表都没有了。  原因：hive可以管理hdfs上的文件，用表的形式来管理文件数据。  而表名、表里有哪些字段，字段类型、哪张表存在哪个数据下等这些表信息，称之为hive的元数据信息。  知识点：hive的元数据信息不是存在hdfs上的，而是存在hive自带的derby关系型数据库里的。  即hive管理的数据是在hdfs上的，hive的元数据信息是存在关系型数据库里的。    上述问题出现的原因就是由于derby数据库引起的，这个数据库功能不完善，仅用于测试。  derby数据库存储hive元数据的方式：当在bin目录下进入hive时，derby数据会在bin目录下生成一个metastore\_db目录，将元数据信息存在这个目录下。  当换个目录，比如在home 目录下hive时，derby又会在home目录下生成一个metastore\_db目录，存储元数据信息。    解决办法：将默认使用的derby数据库换成mysql数据库 |

**linux下的mysql安装**

Hive的数据，是存在HDFS里的。

元数据信息(hive有哪些数据库，每个数据库有哪些表等这类信息)存在关系型数据库里，hive默认用的是derby数据库来存储。即hive工作时，除了要依赖Hadoop，还要依赖关系型数据库。

注意：虽然我们能通过HDFS查看到hive有哪些数据库，有哪些表，以及表里的数据，但是，这不是元数据信息。HDFS最主要的是存储hive的数据信息。

之前遇到的问题是：当退出后，切换到不同的目录来进入hive，发现库和表没有了，是因为，第一次从bin目录进入hive,会在bin目录下创建一个metastore.db目录，在这个目录下，创建一个derby.log文件来存储元数据信息。这个元数据信息是基于bin目录来创建的。而切换到其他目录进入hive时，查询时不是基于bin目录来查询的，所有查不到元数据信息，导致查不到。

这个问题是derby数据库本身的问题，所以，我们不能用derby数据库，此外，用derby数据库，也不支持并发，比如一个人在操作hive，如果此时有其他人想用hive，用不了。所以我们选择用mysql数据库。目前hive支持derby和mysql两种数据库。

mysqld 是服务端程序

mysql是命令行客户端程序

**hive的mysql安装配置**

hive-site.xml

权限分配

grant all privileges on \*.\* to 'root'@'hadoop01' identified by 'root' with grant option;

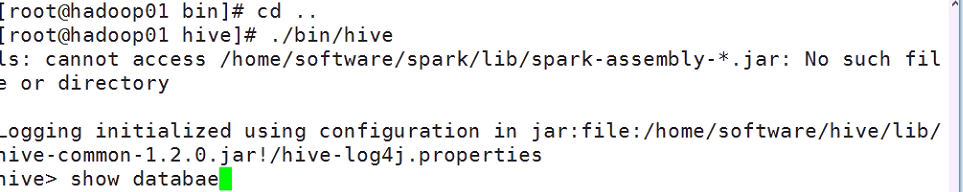
以及

flush privileges;

load data local inpath '/home/1.txt' into table stu;

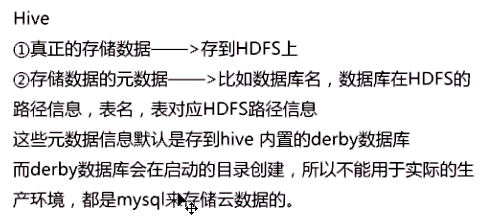
create table stu(id int,name String) row format delimited fields terminated by ' ';

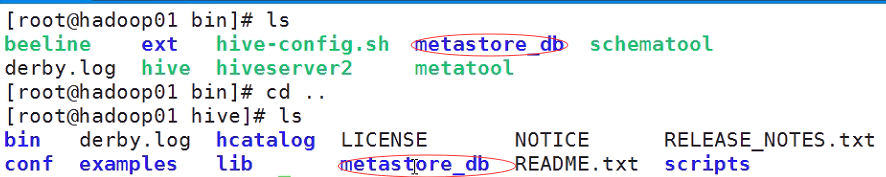
insert overwrite table stu2 select \* from stu;



hive/bin#sh hive

hive#./bin/hive ---

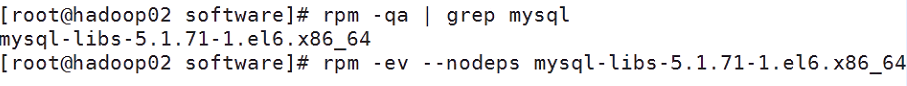




安装mysql:删除内置的mysql

rpm -qa |grep mysql

rpm -ev --nodeps +名字

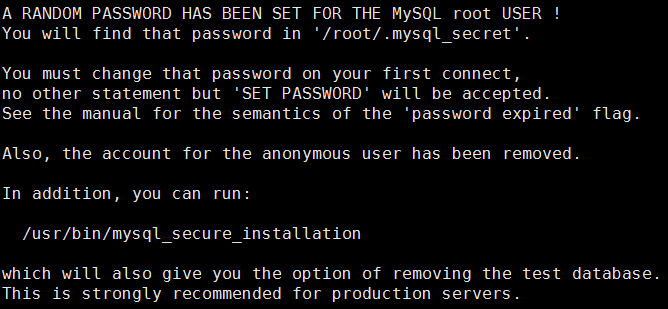


创建mysql用户

groupadd mysql

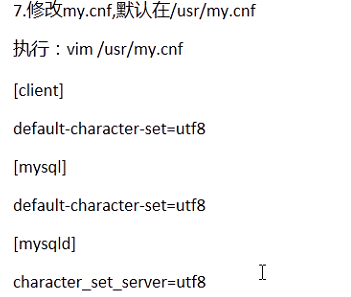
useradd -r -g mysql mysql

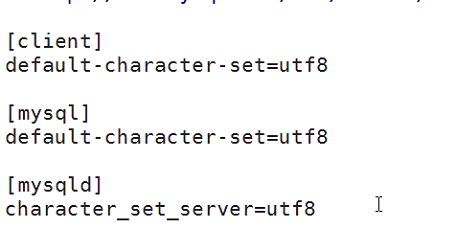
rpm -ivh My-server



rpm -client

修改,vim /usr/my.cnf





**hive的内部表和外部表**

**内部表的概念**

先在hive里建一张表，然后向这个表插入数据（用insert可以插入数据，也可以通过加载外部文件方式来插入数据),这样的表称之为hive的内部表。

**外部表的概念**

HDFS里已经有数据了，比如有一个2.txt文件，里面存储了这样的一些数据,然后，通过hive创建一张表stu来管理这个文件数据。则stu这样表称之为外部表。

create external table stu1(id int,name string)row format delimited fields terminated by ' ' location '/info';

hive无论是内部表或外部表，当向HDFS对应的目录节点下追加文件时（只要格式符合），hive都可以把数据管理进来

**区别**:通过hive执行：drop table stu 。如果stu是一个内部表，则HDFS对应的目录节点会被删除。如果stu是一个外部表，HDFS对应的目录节点不会删除.

**\*\*Hive分区表**

**概念**

Hive也支持分区表,对数据进行分区可以提高查询时的效率。

普通表和分区表区别：有大量数据增加的需要建分区表

**语法**

**执行：**create table book (id int, name string) partitioned by (category string) row format delimited fields terminated by '\t';

**注：**在创建分区表时，partitioned字段可以不在字段列表中。生成的表中自动就会具有该字段。category 是自定义的字段。

**分区表加载数据**

1）load data local inpath '/home/cn.txt' overwrite into table book partition (category='cn');

**分区命令**

**1.显示分区**

show partitions iteblog;

**2.添加分区**

alter table book add partition (category='jp') location '/user/hive/warehouse/test.db/book/category=jp';

或者：

msck repair table book;

**3.删除分区**

alter table book drop partition(category='cn')

**4.修改分区**

alter table book partition(category='french') rename to partition (category='hh');

**hive数据类型**

**基本类型**:int boolean float double string

**复杂类型**:array map struct

**1.数组类型**

a1--100,200,300

create external table a1(info array<int>) row format delimited fields terminated by '\t' collection items terminated by ',' location '/a1';

a2---100,200,300 tom,jary

create external table a2(info1 array<int>,info2 array<string>) row format delimited fields terminated by '\t' collection items terminated by ',' location '/a2';

**2.map类型**

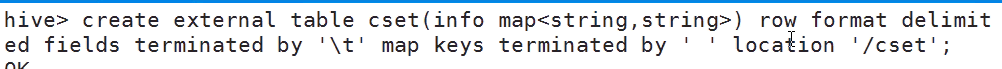
m1

create external table m1(info map<string,int>) row format delimited fields terminated by '\t' map keys terminated by ',' location '/m1';

查询:select info ['tom'] from m1;

select info ['tom'] from m1 where info ['tom']is not null;

cset;--tom 192.168.234.21



**3.strut类型 --**tom 23

create external table s1(info struct<name:string,age:int>) row format delimited fields terminated by '\t' collection items terminated by ',' location '/m1';

select info.age from s1;

**Hive explode**-explode 命令可以将行数据，按指定规则切分出多行。

**word**--hello hive

create external table word(words string) row format delimited fields terminated by '\t' location '/word';

select explode (split(words,' ')) from word;

**hive常用字符串操作函数**

|  |  |  |
| --- | --- | --- |
| 返回类型 | 函数名 | 描述 |
| int | length(string A) | 返回字符串A的长度  select length(weoirjewo);  select length(name) from stu;  此函数在实际工作，可以用于校验手机号，身份号等信息的合法性 |
| string | reverse(string A) | 返回字符串A的反转结果  select reverse('abcd');  select length(name) from stu; |
| string | concat(string A, string B…) | 字符串连接函数  select concat ('a','b');  select concat(id,name) from stu;  select concat(id,',',name) from stu;    http://  [www.baidu.com](http://www.baidu.com)  ?get |
| string | concat\_ws(string SEP, string A, string B…) | 带分隔符字符串连接函数：concat\_ws  select concat\_ws('.','www','baidu','com');  //www.baidu.com |
| string | substr | substr,substring  select substr('abcde',2);从第二个截，截到结尾  select substr('abcde',1,3);从第一个截，截三个长度  select substr('wfeww',-2);从尾部截，截两个长度  可以用于比如截取身份证后几位操作 |
| string | upper(string a)  ucase(string a) | 转大写 |
| string | lower(string a)  lcase(string a) | 转小写 |
| string | trim(string a) | 去空格  select trim (' fwoei '); |
| string | ltrim(string a) | 左边去空格函数 |
| string | rtrim(string a) | 右边去空格函数 |
| string | **regexp\_replace**(string A, string B, string C) | 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,  对需要转义的字符，用[]，比如[\*]，类似oracle中的regexp\_replace函数。 |
| string | **regexp\_extract**(string subject, string pattern, int index) | 将字符串subject按照pattern正则表达式的规则拆分，返回index指定的字符  select regexp\_extract('foothebar', 'foo(.\*)(bar)', 1)； //the   select regexp\_extract('foothebar', 'foo(.\*)(bar)', 2)；//bar  select regexp\_extract('foothebar', 'foo(.\*)(bar)', 0)；//全取 foothebar |
| string | repeat(string str, int n) | 返回重复n次后的str字符串  select repeat('abc',5) |
| array | split(string str, string pat) | 分割字符串函数: split  按照pat字符串分割str，会返回分割后的字符串数组  select split('abtcdtef','t')；  ["ab","cd","ef"] |

 length::

select length ('fwefewf');-->7

select length(words) from word;

10

12

10

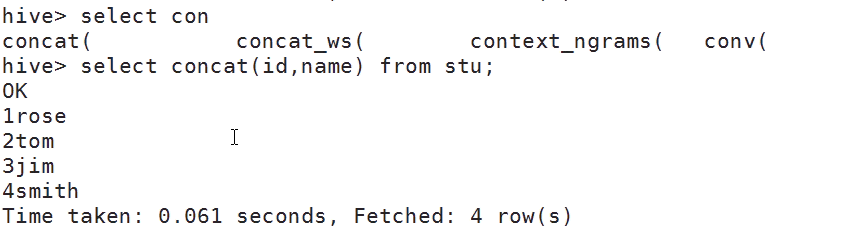
10

10

reverse(String A)反转

concat

concat\_ws



**sbustr**

**HIve的UDF**

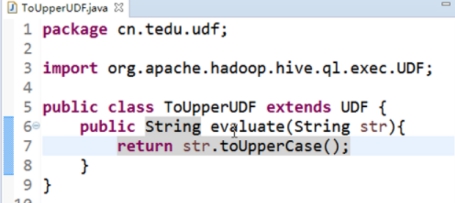
如果hive的内置函数不够用，我们也可以自己定义函数来使用，这样的函数称为hive的用户自定义函数，简称UDF。

**实现步骤：**

1.新建java工程，导入hive相关包，导入hive相关的lib。

2.创建类继承UDF

3.自己编写一个evaluate方法，返回值和参数任意。



4.将写好的类打成jar包，上传到linux中

5.在hive命令行下，向hive注册UDF：add jar /xxxx/xxxx.jar

6.create temporary function fname as 'cn.tedu.ToUpperUDF';

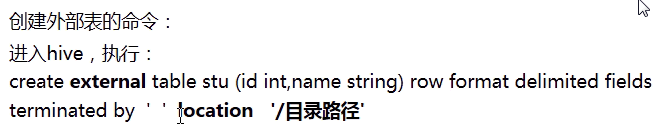
7.select fname('dfdjsf');

---------转换为大写,测试成功!

Big\_day15

Hive02

复习



**作业:**

word1 --hello hive

create external table word(words string) location '/word';

select explode(split(words,' ')) from word;

select count(\*) from (select explode(split(words,' '))a1 from word) w1 group by w1.a1;

作业2,正则替换 --@@@http://www.ali.com:192.16@@@8.127.23:123

r1

create external table r1(info string) location '/r1';

select regexp\_replace(info,'[\*|&|%|@|#]','')from r1;

select regexp\_replace(regexp\_replace(info,'[\*|&|%|@|#]',''),'[.][.]','.')from r1;

作业三:正则提取 --<http://www.baidu.com:192.168.234.22:312>

create external table e1(info string) location '/e1';

select regexp\_extract(info,'http://(.\*)(:)(.\*)(:)(.\*)',1) from e1;

去掉[www.com](http://www.com)

select \* from(select regexp\_extract(info,'http://(.\*)(:)(.\*)(:)(.\*)',1)a1 from e1)w1 where size(split(w1.a1,'[.]'))==3;

**hive的join操作**

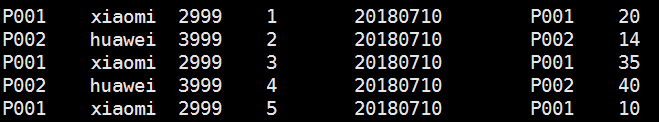
1）create external table order\_t (id string,time string,pid string,amount int) row format delimited fields terminated by ' ' location '/order';

2）create external table product\_t (pid string,name string,price int) row format delimited fields terminated by ' ' location '/product';

**查询：**

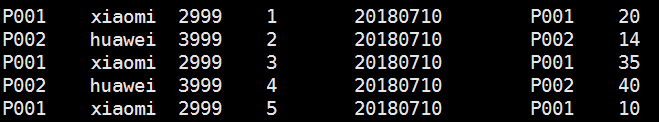
**join,默认inner**

select \* from product\_t join order\_t on product\_t.pid=order\_t.pid;



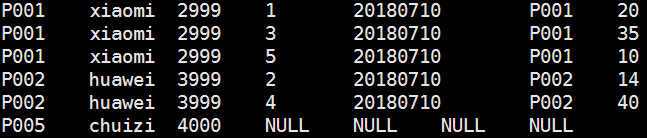
**inner join**

select \* from product\_t inner join order\_t on product\_t.pid=order\_t.pid;



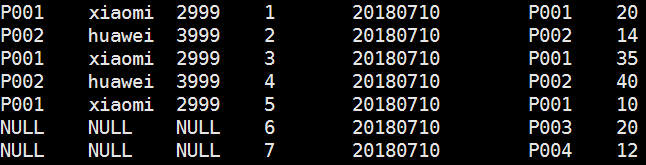
**left join**

select \* from product\_t left join order\_t on product\_t.pid=order\_t.pid;



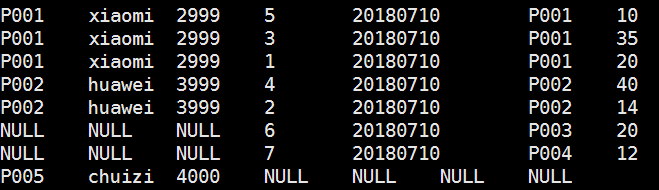
**right join**

select \* from product\_t right join order\_t on product\_t.pid=order\_t.pid;



**Full outer join**

select \* from product\_t full outer join order\_t on product\_t.pid=order\_t.pid;



**left semi join**

select \* from product\_t left semi  join order\_t on product\_t.pid=order\_t.pid;

这种join解决的是exist in（是否存在） 的问题

a表里哪些数据在b表中出现过

IMG_262

MapReduce的join带来数据倾斜

解决---map side join

**Hive 解决数据倾斜问题**

**概述**

什么是数据倾斜以及数据倾斜是怎么产生的？

简单来说数据倾斜就是数据的key 的分化严重不均，造成一部分数据很多，一部分数据很少的局面。

举个 word count 的入门例子，它的map 阶段就是形成 （“aaa”,1）的形式，然后在reduce 阶段进行 value 相加，得出 “aaa” 出现的次数。若进行 word count 的文本有100G，其中 80G 全部是 “aaa” 剩下 20G 是其余单词，那就会形成 80G 的数据量交给一个 reduce 进行相加，其余 20G 根据 key 不同分散到不同 reduce 进行相加的情况。如此就造成了数据倾斜，临床反应就是 reduce 跑到 99%然后一直在原地等着 那80G 的reduce 跑完。

 如此一来 80G 的 aaa 将发往同一个 reducer ，由此就可以知道 reduce 最后 1% 的工作在等什么了。

**为什么说数据倾斜与业务逻辑和数据量有关？**

从另外角度看数据倾斜，其本质还是在单台节点在执行那一部分数据reduce任务的时候，由于数据量大，跑不动，造成任务卡住。若是这台节点机器内存够大，CPU、网络等资源充足，跑 80G 左右的数据量和跑10M 数据量所耗时间不是很大差距，那么也就不存在问题，倾斜就倾斜吧，反正机器跑的动。所以机器配置和数据量存在一个合理的比例，一旦数据量远超机器的极限，那么不管每个key的数据如何分布，总会有一个key的数据量超出机器的能力，造成 reduce 缓慢甚至卡顿。

业务逻辑造成的数据倾斜会多很多，日常使用过程中，容易造成数据倾斜的**原因**可以归纳为几点：

**1）group by**

**2）distinct count(distinct xx)**

**3）join**

**如何处理group by的数据倾斜问题**

1、调优参数

hive> set hive.groupby.skewindata=true;

  数据倾斜时负载均衡，当选项设定为true，**生成的查询计划会有两个MRJob**。第一个MRJob 中，Map的输出结果集合会随机分布到Reduce中，每个Reduce做部分聚合操作，并输出结果，这样处理的结果是相同的GroupBy Key有可能被分发到不同的Reduce中，从而达到负载均衡的目的；第二个MRJob再根据预处理的数据结果按照GroupBy Key分布到Reduce中（这个过程可以保证相同的GroupBy Key被分布到同一个Reduce中），最后完成最终的聚合操作。

由上面可以看出起到至关重要的作用的其实是第二个参数的设置，它使计算变成了两个mapreduce，先在第一个中在 shuffle 过程 partition 时随机给 key 打标记，使每个key 随机均匀分布到各个 reduce 上计算，但是这样只能完成部分计算，因为相同key没有分配到相同reduce上，所以需要第二次的mapreduce,这次就回归正常 shuffle,但是数据分布不均匀的问题在第一次mapreduce已经有了很大的改善，因此基本解决数据倾斜。

参数是:会话级别生效,灵活

**\*\*hive优化**

**&1）map side join**

mapJoin的主要意思就是，当链接的两个表是一个比较小的表和一个特别大的表的时候，我们把比较小的table直接放到内存中去，然后再对比较大的表格进行map操作。join就发生在map操作的时候，每当扫描一个大的table中的数据，就要去去查看小表的数据，哪条与之相符，继而进行连接。**这里的join并不会涉及reduce操作**。map端join的优势就是在于没有shuffle，在实际的应用中，我们这样设置：

set hive.auto.convert.join=true;

此外，hive有一个参数：hive.mapjoin.smalltable.filesize，默认值是25mb（其中一个表大小小于25mb时，自动启用mapjoin）

**2）join语句优化--**先查询再join

优化前

select m.cid,u.id form order m join customer u on m.cid=u.id where m.dt=’20160801’;

优化后

select m.cid,u.id from (select cid from order where dt=’20160801’)m join customer u on m.cid = u.id

**注意：**Hive在做join时，小表写在前（左边）。

**3）group by 优化**

hive.groupby.skewindata=true 参见调优参数

如果group by过程出现倾斜，应该设置为true

**4）count distinct 优化**

优化前

select count(distinct id )from tablename

优化后

select count(\*) from (select distinct id from tablename)tmp;

 去重

案例:--100 200 300 400 200 300 100

create external table distinct1(info string) location '/distinct';

设置reduce数量

set mapred.reduce.tasks=3;

select distinct(info) from distinct1;----number of reducers:3 已经去重了

去重后再计数

select count(distinct(info)) from distinct1; ---reduce又变为1,负载过高

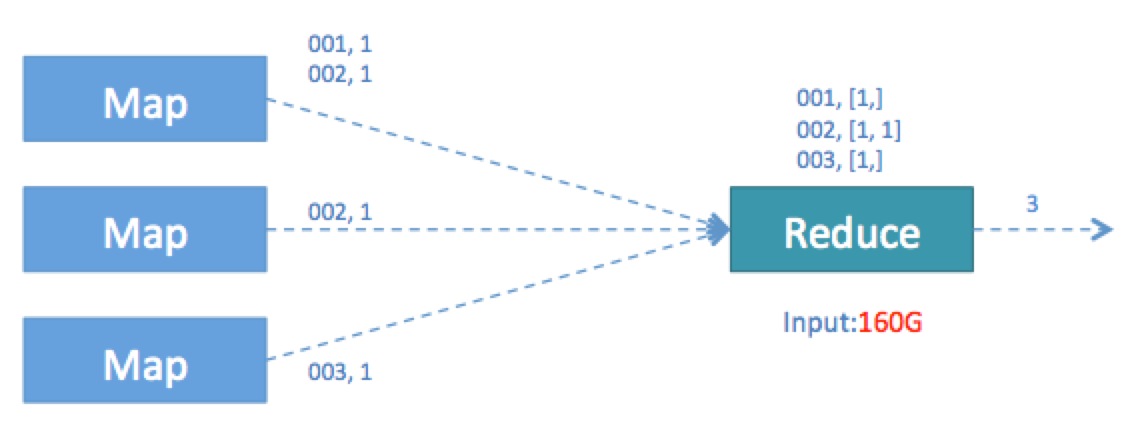
先是一个reduce,后全局何为一个 Total jobs = 2 Launching Job 1 out of 2

日常统计场景中，我们经常会对一段时期内的字段进行消重并统计数量，SQL语句类似于

SELECT COUNT( DISTINCT id ) FROM TABLE\_NAME WHERE ...;

这条语句是从一个表的符合WHERE条件的记录中统计不重复的id的总数。

该语句转化为MapReduce作业后执行示意图如下，图中还列出了我们实验作业中Reduce阶段的数据规模：



由于引入了DISTINCT，因此在Map阶段无法利用combine对输出结果消重，必须将id作为Key输出，在Reduce阶段再对来自于不同Map Task、相同Key的结果进行消重，计入最终统计值。

我们看到作业运行时的Reduce Task个数为1，对于统计大数据量时，这会导致最终Map的全部输出由单个的ReduceTask处理。这唯一的Reduce Task需要Shuffle大量的数据，并且进行排序聚合等处理，这使得它成为整个作业的IO和运算瓶颈。

经过上述分析后，我们尝试显式地增大Reduce Task个数来提高Reduce阶段的并发，使每一个Reduce Task的数据处理量控制在2G左右。具体设置如下：

set mapred.reduce.tasks=100

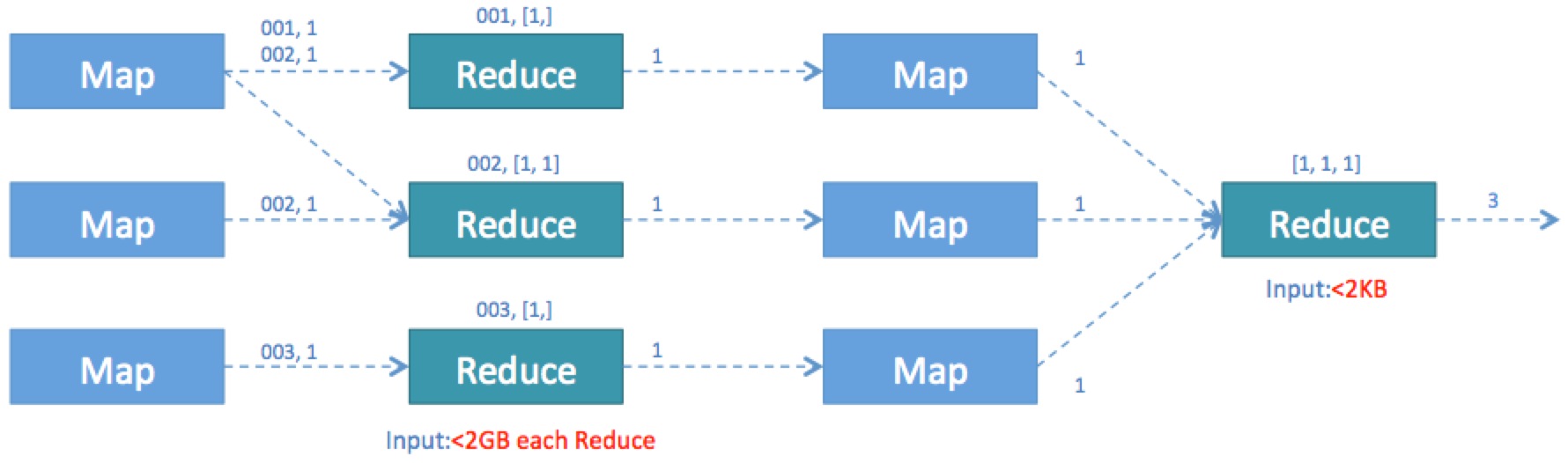
调整后我们发现这一参数并没有影响实际Reduce Task个数，Hive运行时输出“Number of reduce tasks determined at compile time: 1”。

原因是Hive在处理COUNT这种“全聚合(full aggregates)”计算时，它会忽略用户指定的Reduce Task数，而强制使用1。

所以我们只能采用变通的方法来绕过这一限制。我们利用Hive对嵌套语句的支持，**将原来一个MapReduce作业转换为两个作业**，在第一阶段选出全部的非重复id，在第二阶段再对这些已消重的id进行计数。这样在第一阶段我们可以通过增大Reduce的并发数，并发处理Map输出。在第二阶段，由于id已经消重，因此COUNT(\*)操作在Map阶段不需要输出原id数据，只输出一个合并后的计数即可。这样即使第二阶段Hive强制指定一个Reduce Task，极少量的Map输出数据也不会使单一的Reduce Task成为瓶颈。改进后的SQL语句如下：

SELECT COUNT(\*) FROM (SELECT DISTINCT id FROM TABLE\_NAME WHERE … ) t;

这一优化使得在同样的运行环境下，优化后的语句执行只需要原语句20%左右的时间。优化后的MapReduce作业流如下：

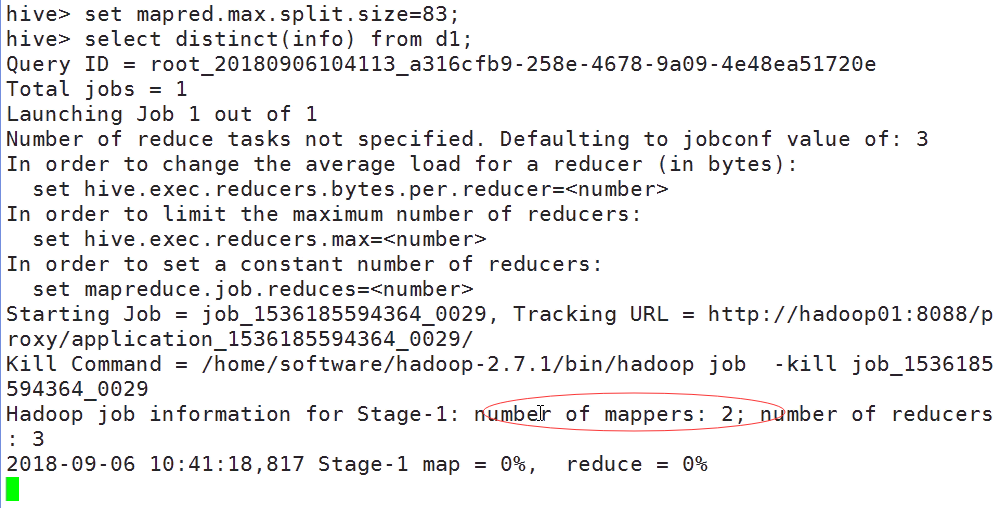


**5）调整切片数（map任务数）**

Hive底层自动对小文件做了优化，用了CombineTextInputFormat，将做个小文件切片合成一个切片。合成完之后的切片大小，如果>mapred.max.split.size 的大小，就会生成一个新的切片。

mapred.max.split.size 默认是128MB

set mapred.max.split.size=134217728（128MB)



**6）JVM重利用**

set mapred.job.reuse.jvm.num.tasks=20(默认是1个）

JVM重用是hadoop调优参数的内容，对hive的性能具有非常大的影响，特别是对于很难避免小文件的场景或者task特别多的场景，这类场景大多数执行时间都很短。这时JVM的启动过程可能会造成相当大的开销，尤其是执行的job包含有成千上万个task任务的情况。

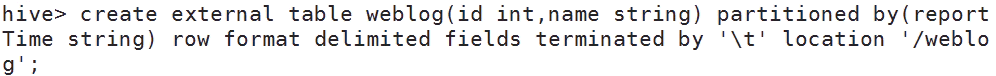
JVM重用可以使得一个JVM进程在同一个JOB中重新使用N次后才会销毁。

**7）启用严格模式**

在hive里面可以通过严格模式防止用户执行那些可能产生意想不到的不好的效果的查询,从而保护hive的集群。

IMG_266

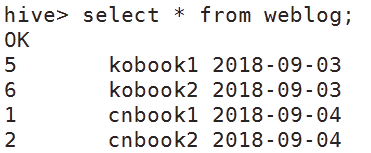
建表



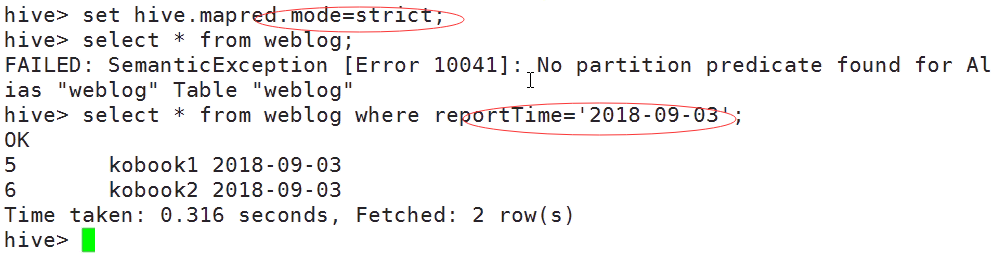
建分区表

IMG_268

查询--当前为分区表



严格模型,分区模式跟上分区字段



严格模式排序要加上limit,orderby排序是全局模式

退出严格模式

IMG_271

用户可以通过 set hive.mapred.mode=strict 来设置严格模式，改成unstrict则为非严格模式。

在严格模式下，用户在运行如下query的时候会报错：

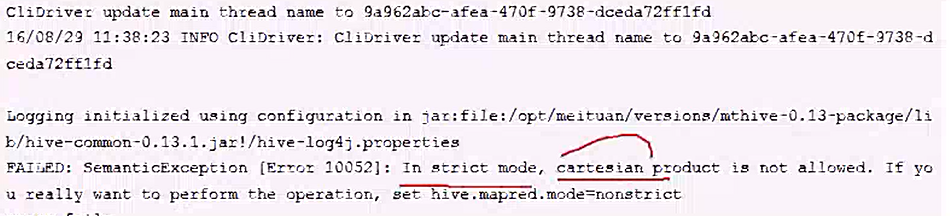
①分区表的查询没有使用分区字段来限制

②使用了order by 但没有使用limit语句。（如果不使用limit，会对查询结果进行全局排序，消耗时间长）

③产生了笛卡尔积

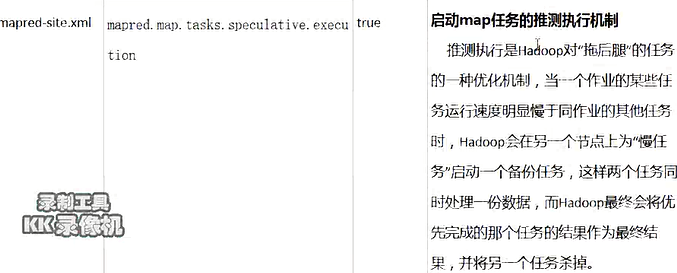
当用户写代码将表的别名写错的时候会引起笛卡尔积，例如

SELECT \*FROM origindb.promotion\_\_campaign c JOIN origindb.promotion\_\_campaignex ce ON c.id = c.id limit 1000



**8）关闭推测执行**

"拖后腿"hadoop中



因为在测试环境下我们都把应用程序跑通了，如果还加上推测执行，如果有一个数据分片本来就会发生数据倾斜，执行执行时间就是比其他的时间长，那么hive就会把这个执行时间长的job当作运行失败，继而又产生一个相同的job去运行，后果可想而知。可通过如下设置关闭推测执行：

set mapreduce.map.speculative=false

set mapreduce.reduce.speculative=false

set hive.mapred.reduce.tasks.speculative.execution=false

Hive的分桶表

**如何使用分桶表**

1.创建带桶的 table ：

create table teacher(class string,name string) clustered by (class) into 3 buckets row format delimited fields terminated by ' ';

2.开启分桶机制：

set hive.enforce.bucketing=true;

3.往表中插入数据：

insert overwrite table teacher select \* from tmp;//需要提前准备好temp，从temp查询数据写入到teacher

**注：**teacher是一个分桶表，对于分桶表，不允许以外部文件方式导入数据，只能从另外一张表数据导入。

 分桶表是内部表

**temp文件数据样例：**

java zhang

web wang

java zhao

java qin

web liu

web zheng

ios li

linux chen

ios yang

ios duan

linux ma

linux xu

java wen

web wu

**作用及原理**

分桶的原理是根据指定的列的计算hash值模余分桶数量后将数据分开存放。方便数据抽样

**注：分桶语法—TABLESAMPLE(BUCKET x OUT OF y)**

y必须是table总bucket数的倍数或者因子。hive根据y的大小，决定抽样的比例。抽样,数据少的时候,可能不准确

**例如：**table总共分了3份，当y=3时，抽取(3/3=)1个bucket的数据，当y=6时，抽取(3/6=)1/2个bucket的数据。

x表示从哪个bucket开始抽取。

**例如：**table总bucket数为3，tablesample(bucket 3 out of 3)，表示总共抽取（3/3=）1个bucket的数据，抽取第3个bucket的数据。

**再例如：**table总bucket数为32，tablesample(bucket 3 out of 16)，表示总共抽取（32/16=）2个bucket的数据，分别为第3个bucket和第（3+16=）19个bucket的数据。

查询第一个桶里数据，并返回一半的数据：

select \* from bucketed\_user tablesample(bucket 1 out of 6 on id);

 Sqoop安装及指令

**Sqoop介绍**

sqoop是Apache 提供的工具,可用于hdfs和关系型数据库之间数据的导入和导入,可以从hdfs导出数据到关系型数据库，也可以从关系型数据库导入数据到hdfs。

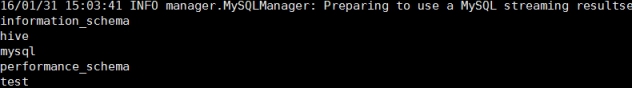
**实现步骤：**

2..需要将要**连接的数据库的驱动包**加入sqoop的lib目录下（本例中用的是mysql数据库）

3.利用指令操作sqoop

**Sqoop基础指令（在Sqoop的bin目录下执行下列指令）**

|  |  |
| --- | --- |
| 说明 | 指令示例 |
| 查看mysql所有数据库 | sh sqoop list-databases --connect  jdbc:mysql://192.168.150.138:3306/ -username root -password root |
| 查看指定数据库下的所有表 | sh sqoop list-tables --connect jdbc:mysql://hadoop02:3306/hive -username root -password root |
| 关系型数据库==>hdfs  sh sqoop import -help（查看import的帮助指令） | **实现步骤：**  1.现在mysql数据库的test数据下建立一张tabx表，并插入测试数据  建表：create table tabx (id int,name varchar(20));  插入：insert into tabx (id,name) values (1,'aaa'),(2,'bbb'),(3,'ccc')，(1,'ddd'),(2,'eee'),(3,'fff');  2.进入到sqoop的bin目录下，执行导入语句  导入：  sh sqoop import --connect jdbc:mysql://192.168.150.138:3306/test --username root --password root --table tabx --target-dir '/sqoop/tabx' --fields-terminated-by '|' -m 1; |
| hdfs==>关系型数据库 | 执行：sh sqoop export --connect jdbc:mysql://192.168.150.138:3306/test --username root --password root --export-dir '/sqoop/tabx/part-m-00000' --table taby -m 1 --fields-terminated-by '|'  注：sqoop只能导出数据，不能自动建表。所以在导出之前，要现在mysql数据库里建好对应的表 |

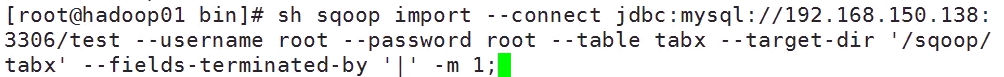


sh sqoop list-databases --connect jdbc:mysql://176.129.8.81:3306/ -username root -password root

sh sqoop list-tables --connect jdbc:mysql://176.129.8.81:3306/hive -username root -password root

IMG_275

insert into tabx (id,name) values (1,'aaa'),(1,'bbb'),(1,'ccc'),(1,'ddd'),(1,'eee'),(1,'fff'),



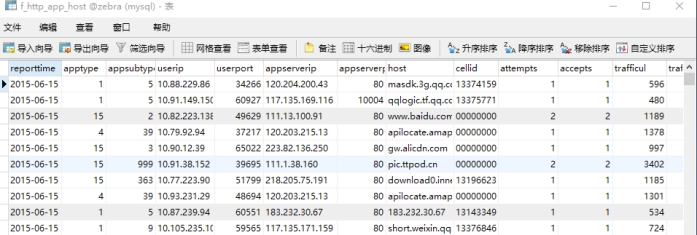


sh sqoop export --connect jdbc:mysql://176.129.8.81:3306/test --username root --export-dir '/user/hive/'

Zebra业务回顾--电信流量

**zebra业务回顾**

zebra项目最开始阶段会对日志文件进行分析统计，针对apptype,userip等20个字段做了统计，然后把最后的结果落地到数据库里。这张表相当于总表 （f\_http\_app\_host）



在企业里做到这步并没有结束，因为后续还要做数据分析，可能会针对此表进行多个维度的查询和统计，比如：

1.应用欢迎度

2.各网站表现

3.小区Http上网能力

4.小区上网洗好

所以我们可以根据以上四个维度，建立对应的表，并从f\_http\_app\_host 取出对应的数据，然后做统计，最后交给前端做数据可视化的工作。比如下图是针对应用受欢迎程度的数据可视化图：

下图展示了前10名最受欢迎应用，是根据每个应用产生的总量来统计的（一般来说，流量越大，用户越多）



**建表语句：**

create table D\_H\_HTTP\_APPTYPE(

hourid datetime,apptype int,appsubtype int,attempts bigint,accepts bigint,succratio double,trafficul bigint,trafficdl bigint,totaltraffic bigint,retranul bigint,retrandl bigint,

retrantraffic bigint,failcount bigint,transdelay bigint);

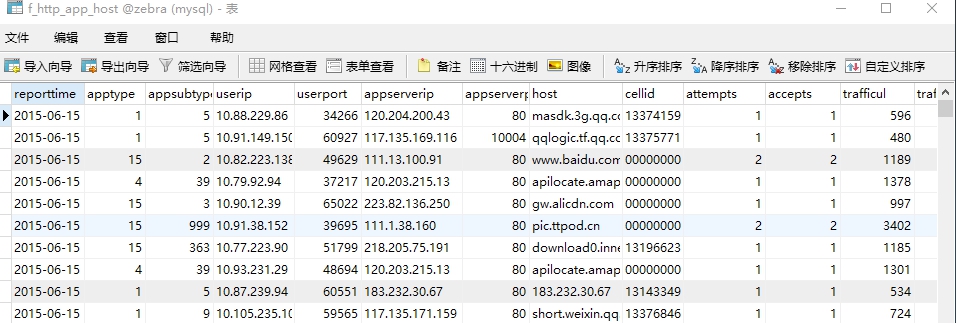
业务说明

**数据以 | 分割后，每个数据的含义（仅展示项目里用到的字段数据）**

|  |  |  |  |
| --- | --- | --- | --- |
| **下标位置** | **字段标识** | **数据类型** | **字段释义** |
| 16 | Cell ID | byte | UE所在小区的ECI |
| 18 | App Type Code | byte | 业务类型编码，参见附录D XDR类型编码定义 |
| 23 | App Sub-type | byte | 应用小类  根据集团定义的识别规则识别出来的小类, 参见《中国移动数据流量DPI识别能力规范》。  集团未定义的各厂家根据自己的DPI进行识别 |
| 26 | USER\_IP | byte | 终端用户的IPv4地址，如无则填全F |
| 28 | User Port | byte | 用户的四层端口号 |
| 30 | App Server IP | byte | 访问服务器的IPv4地址，如无则填全F |
| 32 | App Server Port | byte | 访问的服务器的端口 |
| 58 | HOST | char | 访问域名 |
| 19 | ProcdureStartTime | long | 请求起始时间 |
| 20 | ProcdureEndTime | long | 请求结束时间 |
| 22 | App Type | byte | 应用大类更多信息参见《中国移动数据流量DPI识别能力规范》 |
| 33 | UL Data | byte | 上行流量 |
| 34 | DL Data | byte | 下行流量 |
| 39 | RetranUL | byte | 上行TCP重传报文数 |
| 40 | RetranDL | byte | 下行TCP重传报文数 |
| 54 | HTTP/WAP事务状态 | byte | HTTP/WAP2.0层的响应码，参见附录A 状态编码 |

**五、zebra业务说明**

zebra项目最开始阶段会对日志文件进行分析统计，然后把最后的结果落地到数据库里。



建表语句

create table F\_HTTP\_APP\_HOST(

reporttime datetime,

...

);

后期可能会根据统计出来的数据，进行业务拆分。形成几个不同的维度进行查询：

1.应用欢迎度

2.各网站表现

3.小区Http上网能力

4.小区上网洗好

**应用欢迎度表说明**

|  |  |  |  |
| --- | --- | --- | --- |
| D\_H\_HTTP\_APPTYPE(应用欢迎度) |  |  |  |
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | appType | int | 应用大类 |
| 2 | appSubtype | int | 应用小类 |
| 3 | attempts | int(20) | 尝试次数 |
| 4 | accepts | int(20) | 接受次数 |
| 5 | succRatio | int(20) | 尝试成功率 |
| 6 | trafficUL | int(20) | 上行流量 |
| 7 | trafficDL | int(20) | 下行流量 |
| 8 | totalTraffic | int(20) | 总流量 |
| 9 | retranUL | int(20) | 重传上行报文数 |
| 10 | retranDL | int(20) | 重传下行报文数 |
| 11 | retranTraffic | int(20) | 重传报文数据 |
| 12 | failCount | int(20) | 延时失败次数 |
| 13 | transDelay | int(20) | 传输时延 |

建表语句：

create table D\_H\_HTTP\_APPTYPE(

hourid datetime,

...

);

**各网站的表现表**

|  |  |  |  |
| --- | --- | --- | --- |
| D\_H\_HTTP\_HOST（各网站的表现） |  |  |  |
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | host | varchar(50) | 域名 |
| 2 | appServerIP | varchar(20) | 服务器IP |
| 3 | attempts | int(20) | 尝试次数 |
| 4 | accepts | int(20) | 接受次数 |
| 5 | succRatio | int(20) | 尝试成功率 |
| 6 | trafficUL | int(20) | 上行流量 |
| 7 | trafficDL | int(20) | 下行流量 |
| 8 | totalTraffic | int(20) | 总流量 |
| 9 | retranUL | int(20) | 重传上行报文数 |
| 10 | retranDL | int(20) | 重传下行报文数 |
| 11 | retranTraffic | int(20) | 重传报文数据 |
| 12 | failCount | int(20) | 延时失败次数 |
| 13 | transDelay | int(20) | 传输时延 |

建表语句：

#创建各网站表现表

create table D\_H\_HTTP\_HOST(

hourid datetime,

...

);

**小区HTTP上网能力表**

|  |  |  |  |
| --- | --- | --- | --- |
| D\_H\_HTTP\_CELLID（小区HTTP上网能力） |  |  |  |
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | cellid | varchar | 小区ID |
| 2 | attempts | int(20) | 尝试次数 |
| 3 | accepts | int(20) | 接受次数 |
| 4 | succRatio | int(20) | 尝试成功率 |
| 5 | trafficUL | int(20) | 上行流量 |
| 6 | trafficDL | int(20) | 下行流量 |
| 7 | totalTraffic | int(20) | 总流量 |
| 8 | retranUL | int(20) | 重传上行报文数 |
| 9 | retranDL | int(20) | 重传下行报文数 |
| 10 | retranTraffic | int(20) | 重传报文数据 |
| 11 | failCount | int(20) | 延时失败次数 |
| 12 | transDelay | int(20) | 传输时延 |

#创建小区HTTP上网能力表

create table D\_H\_HTTP\_CELLID(

hourid datetime,

...

);

**小区上网喜好表**

|  |  |  |  |
| --- | --- | --- | --- |
| D\_H\_HTTP\_CELLID\_HOST（小区上网喜好） |  |  |  |
| 序号 | 字段 | 字段类型 | 描述 |
| 0 | hourid | datetime | 小时时间片 |
| 1 | cellid | varchar | 小区ID |
| 2 | host | varchar(50) | 域名 |
| 3 | attempts | int(20) | 尝试次数 |
| 4 | accepts | int(20) | 接受次数 |
| 5 | succRatio | int(20) | 尝试成功率 |
| 6 | trafficUL | int(20) | 上行流量 |
| 7 | trafficDL | int(20) | 下行流量 |
| 8 | totalTraffic | int(20) | 总流量 |
| 9 | retranUL | int(20) | 重传上行报文数 |
| 10 | retranDL | int(20) | 重传下行报文数 |
| 11 | retranTraffic | int(20) | 重传报文数据 |
| 12 | failCount | int(20) | 延时失败次数 |
| 13 | transDelay | int(20) | 传输时延 |

#创建小区上网喜好表

create table D\_H\_HTTP\_CELLID\_HOST(

hourid datetime,

...

);

web应用展示zebra业务

此应用只针对**应用受欢迎度表（D\_H\_HTTP\_APPTYPE 表）**做统计

**实现步骤：**

1.数据库建表

2.插入测试数据

3.搭建web应用

4.启动tomcat服务器

5.浏览测试 。在浏览器输入：http://localhost:端口号/项目名 即可访问

最后的展示效果：



Hive实现Zebra

**实现流程：**

使用flume收集数据 --> 落地到hdfs系统中 --> 创建hive的外部表管理hdfs中收集到的日志 --> 利用hql处理zebra的业务逻辑 --> 使用sqoop技术将hdfs中处理完成的数据导出到mysql中

**flume组件工作说明：**

flume在收集日志的时候，按天为单位进行收集。hive在处理的时候，按天作为分区条件，继而对每天的日志进行统计分析。最后，hive将统计分析的结果利用sqoop导出到关系型数据库里，然后做数据可视化的相关工作。

对于时间的记录，一种思路是把日志文件名里的日志信息拿出来，第二种思路是flume在收集日志时，将当天的日期记录下来。我们用第二种思路。

**Flume配置**

/home/software/apache-flume-1.6.0-bin/conf #vim zebra.conf

**配置示例：**

a1.sources=r1

a1.channels=c1

a1.sinks=s1

a1.sources.r1.type=spooldir

a1.sources.r1.spoolDir=/home/zebra

a1.sources.r1.interceptors=i1

a1.sources.r1.interceptors.i1.type=timestamp

a1.sinks.s1.type=hdfs

a1.sinks.s1.hdfs.path=hdfs://192.168.150.137:9000/zebra/reportTime=%Y-%m-%d

a1.sinks.s1.hdfs.fileType=DataStream

a1.sinks.s1.hdfs.rollInterval=30

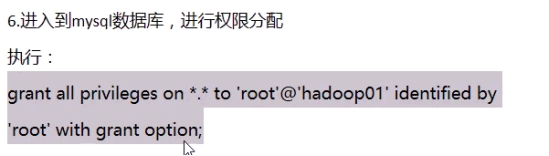
a1.sinks.s1.hdfs.rollSize=0

a1.sinks.s1.hdfs.rollCount=0

a1.channels.c1.type=memory

a1.sources.r1.channels=c1

a1.sinks.s1.channel=c1



将待处理的日志文件上传到/home/zebra下，最终，这个文件会被01虚拟机收集到，最后落到hdfs上。

注意：在上传日志文件的时候，不要在/root/work/data/flumedata 目录下通过rz 上传，因为rz是连续传输文件，这样会使得flume在处理时报错，错误为正在处理的日志文件大小被修改，所以最好是先把日志上传到linux的其他目录下，然后通过mv 指令移动到/root/work/data/flumedata 目录下

**Hive组件工作流程（建表语句不用写，重在了解整个ETL过程，这个过程很重要）**

使用hive，创建zebra数据库

执行：create database zebra;

执行：use zebra;

然后建立分区，再建立表

**详细建表语句：**

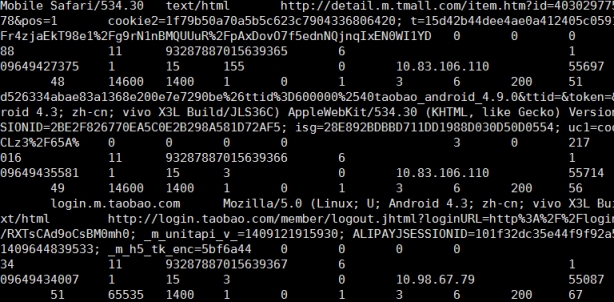
create EXTERNAL table zebra (a1 string,a2 string,a3 string,a4 string,a5 string,a6 string,a7 string,a8 string,a9 string,a10 string,a11 string,a12 string,a13 string,a14 string,a15 string,a16 string,a17 string,a18 string,a19 string,a20 string,a21 string,a22 string,a23 string,a24 string,a25 string,a26 string,a27 string,a28 string,a29 string,a30 string,a31 string,a32 string,a33 string,a34 string,a35 string,a36 string,a37 string,a38 string,a39 string,a40 string,a41 string,a42 string,a43 string,a44 string,a45 string,a46 string,a47 string,a48 string,a49 string,a50 string,a51 string,a52 string,a53 string,a54 string,a55 string,a56 string,a57 string,a58 string,a59 string,a60 string,a61 string,a62 string,a63 string,a64 string,a65 string,a66 string,a67 string,a68 string,a69 string,a70 string,a71 string,a72 string,a73 string,a74 string,a75 string,a76 string,a77 string) partitioned by (reportTime string) row format delimited fields terminated by '|' stored as textfile location '/zebra';

增加分区操作

执行：ALTER TABLE zebra add  PARTITION (reportTime='2018-09-06') location '/zebra/reportTime=2018-09-06';

执行查询，看是否能查出数据

可以通过抽样语法来检验：select \* from zebra TABLESAMPLE (1 ROWS);



清洗数据，从原来的77个字段变为23个字段

**建表语句：**

create table dataclear(reporttime string,appType bigint,appSubtype bigint,userIp string,userPort bigint,appServerIP string,appServerPort bigint,host string,cellid string,appTypeCode bigint,interruptType String,transStatus bigint,trafficUL bigint,trafficDL bigint,retranUL bigint,retranDL bigint,procdureStartTime bigint,procdureEndTime bigint)row format delimited fields terminated by '|';

从zebra表里导出数据到dataclear表里（23个字段的值）

**建表语句：**

insert overwrite table dataclear select concat(reportTime,' ','00:00:00'),a23,a24,a27,a29,a31,a33,a59,a17,a19,a68,a55,a34,a35,a40,a41,a20,a21 from zebra;

处理业务逻辑，得到dataproc表

**建表语句：**

create table dataproc (reporttime string,appType bigint,appSubtype bigint,userIp string,userPort bigint,appServerIP string,appServerPort bigint,host string,cellid string,attempts bigint,accepts bigint,trafficUL bigint,trafficDL bigint,retranUL bigint,retranDL bigint,failCount bigint,transDelay bigint)row format delimited fields terminated by '|';

根据业务规则，做字段处理

**建表语句：**

insert overwrite table dataproc select reporttime,appType,appSubtype,userIp,userPort,appServerIP,appServerPort,host,

if(cellid == '',"000000000",cellid),if(appTypeCode == 103,1,0),if(appTypeCode == 103 and find\_in\_set(transStatus,"10,11,12,13,14,15,32,33,34,35,36,37,38,48,49,50,51,52,53,54,55,199,200,201,202,203,204,205,206,302,304,306")!=0 and interruptType == 0,1,0),if(apptypeCode == 103,trafficUL,0), if(apptypeCode == 103,trafficDL,0), if(apptypeCode == 103,retranUL,0), if(apptypeCode == 103,retranDL,0), if(appTypeCode == 103 and transStatus == 1 and interruptType == 0,1,0),if(appTypeCode == 103, procdureEndTime - procdureStartTime,0) from dataclear;

查询关心的信息，以应用受欢迎程度表为例：

**建表语句：**

create table D\_H\_HTTP\_APPTYPE(hourid string,appType int,appSubtype int,attempts bigint,accepts bigint,succRatio double,trafficUL bigint,trafficDL bigint,totalTraffic bigint,retranUL bigint,retranDL bigint,retranTraffic bigint,failCount bigint,transDelay bigint) row format delimited fields terminated by '|';

根据总表dataproc,按条件做聚合以及字段的累加

**建表语句：**

insert overwrite table D\_H\_HTTP\_APPTYPE select reporttime,apptype,appsubtype,sum(attempts),sum(accepts),round(sum(accepts)/sum(attempts),2),sum(trafficUL),sum(trafficDL),sum(trafficUL)+sum(trafficDL),sum(retranUL),sum(retranDL),sum(retranUL)+sum(retranDL),sum(failCount),sum(transDelay)from dataproc group by reporttime,apptype,appsubtype;

18.查询前5名受欢迎app

select hourid,apptype,sum(totalTraffic) as tt from D\_H\_HTTP\_APPTYPE group by hourid,apptype sort by tt desc limit 5;

**Sqoop组件工作流程：**

将Hive表导出到Mysql数据库中，然后通过web应用程序+echarts做数据可视化工作。

**实现步骤（讲完sqoop后，作为课后作业）：**

1.在mysql建立对应的表

2.利用sqoop导出d\_h\_http\_apptype表

导出语句：

sh sqoop export --connect jdbc:mysql://192.168.150.138:3306/zebra --username root --password root --export-dir '/user/hive/warehouse/zebra.db/d\_h\_http\_apptype/000000\_0' --table D\_H\_HTTP\_APPTYPE -m 1 --fields-terminated-by '|'

Hive JDBC--了解

**Hive的jdbc编程**

hive实现了jdbc接口，所以可以通过java代码操作。但是实际应用中用的不多，一般都是在HDFS储存的文件基础上建立外部表来进行查询处理。所以jdbc了解一下即可。

**实现步骤：**

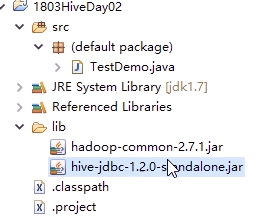
1.在服务器端开启HiveServer服务

./hive --service hiveserver2  &  （以后台线程启动）

2.创建本地工程，导入jar包

导入hive\lib目录下的hive-jdbc-1.2.0-standalone.jar

导入hadoop-2.7.1\share\hadoop\common下的hadoop-common-2.7.1.jar



3.编写jdbc代码执行

代码示例：

 @Test

public void testConnectAndQuery() throws Exception {

//注册数据库驱动，用的hive的jdbc，驱动名固定写死

Class.forName("org.apache.hive.jdbc.HiveDriver");

//如果用的是hive2服务，则写jdbc:hive2，后面跟上hive服务器的ip以及端口号，端口号默认是10000

Connection conn = DriverManager.getConnection("jdbc:hive2://192.168.234.21:10000/park","root","root");

Statement stat = conn.createStatement();

ResultSet rs = stat.executeQuery("select \* from stu");

while(rs.next()){

String name = rs.getString("name");

System.out.println(name);

}

stat.close();

conn.close();

}

1.注册数据库驱动Class.forName("com.mysql.jdbc.Driver");

2.获取数据库连接conn = DriverManager.getConnection("jdbc:mysql:///mydb5", "root", "root");

3.获取传输器stat = conn.createStatement();

4.利用传输器，发送sql到数据库执行，返回执行结果rs = stat.executeQuery("select \* from account");

5.处理结果

while(rs.next()){int id = rs.getInt(1);

String name = rs.getString("name");double money = rs.getDouble("money");

System.out.println(id+name+money);}} catch (Exception e) {e.printStackTrace();throw new RuntimeException();}finally{

6.释放资源

Hive特点

**1.针对海量数据的高性能查询和分析系统**

由于 Hive 的查询是通过 MapReduce 框架实现的，而 MapReduce 本身就是为实现针对海量数据的高性能处理而设计的。所以 Hive 天然就能高效的处理海量数据。

与此同时，Hive 针对 HiveQL 到 MapReduce的翻译进行了大量的优化，从而保证了生成的MapReduce 任务是高效的。在实际应用中，Hive 可以高效的对 TB 甚至 PB级的数据进行处理。

**2.类SQL的查询语言**

HiveQL 和 SQL 非常类似，所以一个熟悉SQL 的用户基本不需要培训就可以非常容易的使用 Hive 进行很复杂的查询。

**3.HiveQL 灵活的可扩展性(Extendibility)**

除了 HiveQL 自身提供的能力，用户还可以自定义其使用的数据类型、也可以用任何语言自定义 mapper 和 reducer 脚本，还可以自定义函数(普通函数、聚集函数)等。这就赋予了 HiveQL 极大的可扩展性。用户可以利用这种可扩展性实现非常复杂的查询。

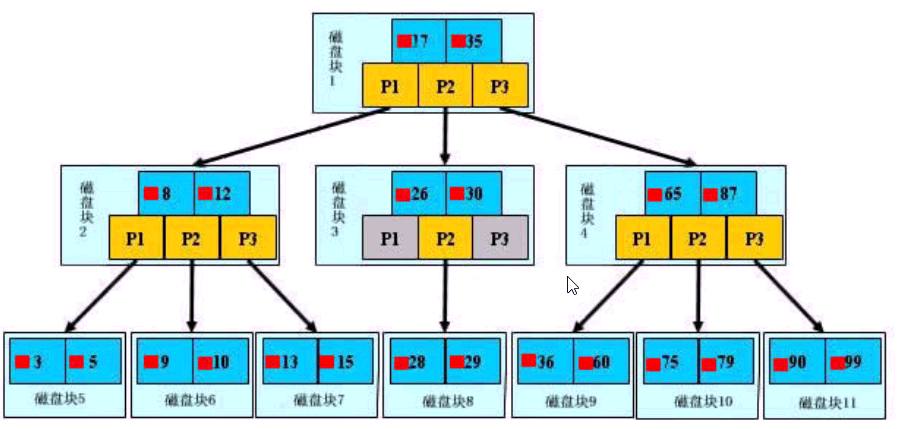
**4.高扩展性(Scalability)和容错性**

Hive本身并没有执行机制，用户查询的执行是通过 MapReduce 框架实现的。由于MapReduce 框架本身具有高度可扩展(计算能力随 Hadoop 机群中机器的数量增加而线性增加)和高容错的特点，所以 Hive也相应具有这些特点。

**5.与 Hadoop 其他产品完全兼容**

Hive 自身并不存储用户数据，而是通过接口访问用户数据。这就使得 Hive支持各种数据源和数据格式。例如，它支持处理 HDFS 上的多种文件格式(TextFile、SequenceFile 等)，还支持处理 HBase 数据库。用户也完全可以实现自己的驱动来增加新的数据源和数据格式。一种理想的应用模型是将数据存储在 HBase 中实现实时访问，而用Hive对HBase 中的数据进行批量分析。

java复习



mysql的底层索引 B+tree

