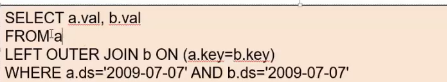
我们看一下这两个sql语句（如下图1所示）

图1：



那么这图1中的两个sql语句有什么区别呢？一个区别就是说一个where放到了外面，一个where条件放到了里面，那么这两个语句哪个会好一点呢？如果是在join的过程中尽量的避免去用where，但是这种情况跟之前我们讲避免用where的思路不太一样了，如果说把你的where条件能够放到你的on里面的话，相当于你在做数据过滤的时候会提前的把这个条件当作一个过滤的条件，过滤一些数据，然后像上面那个语句也就是（下图2所示）

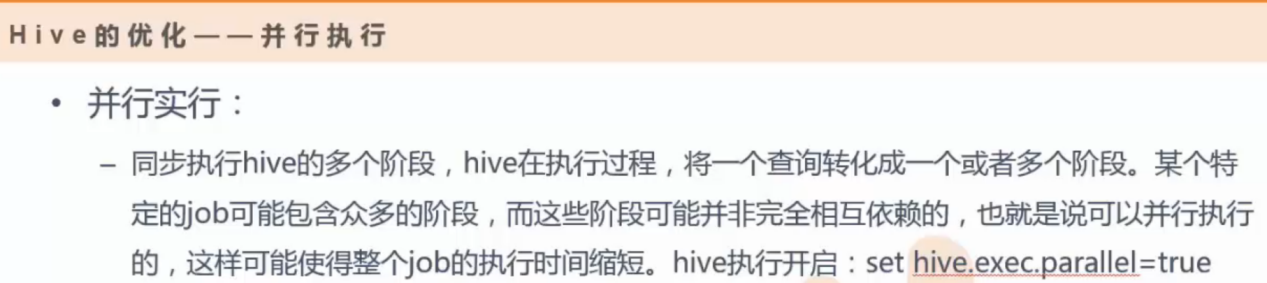
图2：



这个语句已经把前面两个表已经做完了join得到了结果，然后又做了一遍的过滤，相当于把问题分成两个部分，第一步先做join然后再做过滤，但是第二步是我的过滤和join同步执行的。

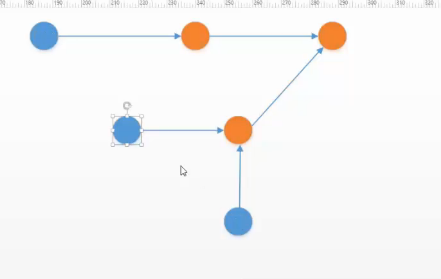
还有一个并行执行，这个是什么意思呢？（如下图3所示）

图3：



我们举个例子：假设说我提交了一个sql，它会生成了一个很多的mapreduce对吧？然后但是每一个节点都代表一个mapreduce（如下图4所示）

图4：



然后这个mapreduce节点之间它还是有一个相互之间的一个依赖关系，标黄色的部分是要依赖于别人的任务，那相当于这几个蓝色的任务相当于前面是没有依赖，而显得这蓝色部分的任务之间是可以相互独立的，那这个时候蓝色的这几个mapreduce是可以并发执行的。

如果想开启这个功能的时候，就要把这个并发的这个参数set hive.exec.parallel=true设置为true，要把那些没有相互依赖之间的关系的一些节点就可以能够尽可能的并行。

然后这块相当于是做了一个总结，但是无论哪种方式其实在某种程度上我们优化的主要的目标大部分问题都是为了解决数据倾斜这么一个问题（如下图5所示）

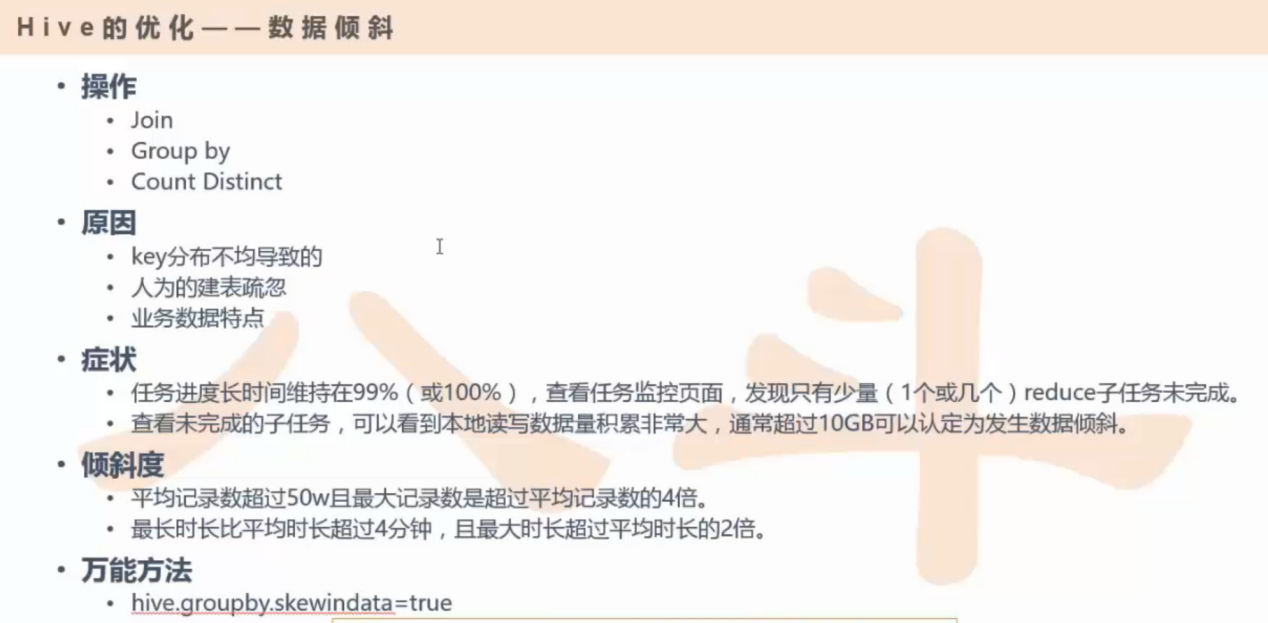
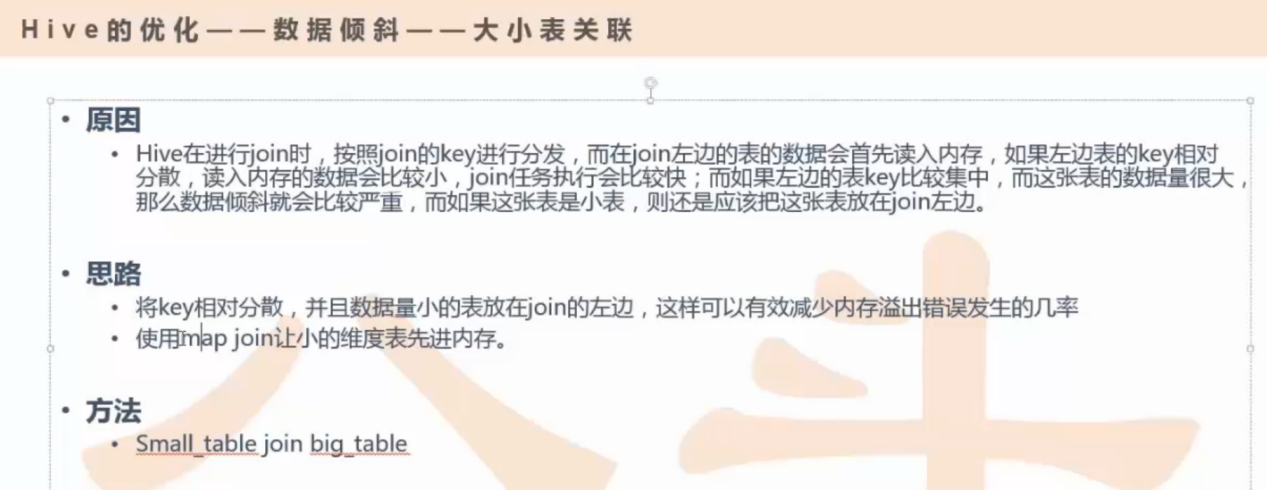
图5：  
 那首先就是一个大表和小表之间的关联，其实我们刚才已经大概的说了一下（如下图6所示）

图6：



那用map join的方式去解决这样的问题，那么到现在其实这个map join这个方法其实也应该很熟悉了是吧？，可以通过刚才所说的那个配置来哪个是小表哪个是大表，然后把这个小表提前进入缓存这样的方式。

还有一种你不通过map join它这个hive本身就可以默认把join的左边就当作一个小表，然后join的右边当作一个大表，所以这是一个大小表关联的一个情况，那如果两张表都是大表怎办？那我们在这里面能不能做一些个优化呢？

假设说我们这个日志里面通常会存在大量的这个user是为空或者为0的情况，就是比如说我们这个日志是在线请求日志，userid代表用户，但是这个用户的信息你不能够保证这个用户的信息都可以完整的记录下来，如果不能够记录下来的话那为了保证他的日志的数据的一致性，那我们会默认对那些无法识别用户我们就给它一个0或者设一个null，所以你会发现那对于整个的一个日志的观察来看，其实这里面会存在大量user=0或者是user=null的情况，但这并不能表明说有0的用户都是代表同一个人这个明白把？因为我没有办法识别那些流量我都会当作0或者都会当作null，所以这个时候这个数据里面会存在大量的空或者大量的0这种情况，那这个时候如果你要对这个userid做一个分桶的时候，0的或者空的这个userid会分到一块，这样导致了某一个桶会变得非常的大，好了那这个时候会必然的导致了你这个数据有过大的倾斜，那怎么样解决这个问题呢？

其实这个问题我们之前大概说过对吧？我们能不能把这个key变成一个字符串，并且加一个随机数，然后把倾斜的数据能够尽可能的分散开来，分散到不同的桶里面去，因为这个user=0和这个user=null是因为我们没有办法识别用户，所以比方说我们要对这个用户做一些数据的关联。

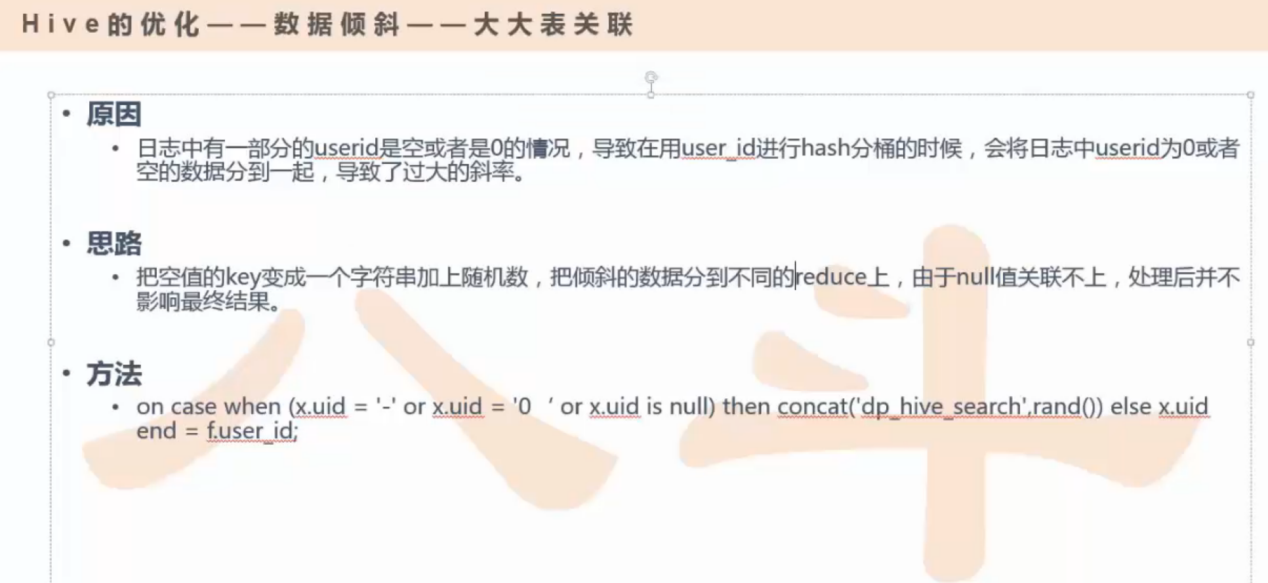
还有一个表，表1用来数据访问，就是用户一个点击日志，表2代表是一个用户基础属性。

那么用户点击日志这个比较好理解是吧？就是判断这个用户是否点击了。用户基础属性是代表这个用户叫什么名字，年龄多少对不对？那我们想对这个两个表做join的话，因为这个表里面存在着大量的等于0和等于null这种情况，那如果说我们要对这个等于0等于null这个情况我们给它一个随机数，在这个字符串后面加一个随机数的话，相当于会把这个用户的一些点击的一些行为会分散到不同的桶里面去，但是你会发现这个用户等于0等于null，它并不代表某一个人，所以它也没有办法确定这个用户的基础属性。

那么这个时候如果你对这个用户userid=0这个，比如说你要把这个key改了，改为一个随机数，那么这个时候以随机数为key，去和用户基础属性表做join的时候你根本就join不上对不对？你join不上的话，相当于从侧面就起到了一个数据过滤的作用，所以它如果代表一个无意义的用户，那么它也没有办法从用户基础属性里面能够查到，那么既然查不到的话，那我也想对这样的一个用户数据进行一个分散开来是吧？

那我就可以完全的对这个key进行随机处理，这样既能实现了负载均衡也能实现了数据过滤，所以这是一列问题，就是如果说这表1和表2这两张表都很大，那么基础属性表也很大，它这个点击日志就会更大了是吧?那这个就跟业务相关了，那比如说这个user=0，我这里面的业务是一个无效的用户，在保不起在哪个系统里面有一个user用户等于0，它就是一个超级用户呢？所以这个就跟你的数据分析也有一定的关系，所以这是大大表关联的一个情况（如下图7所示）

图7：



好了那这个时候，其实这里面也是跟业务相关，这里有一个例子（如下图8所示）

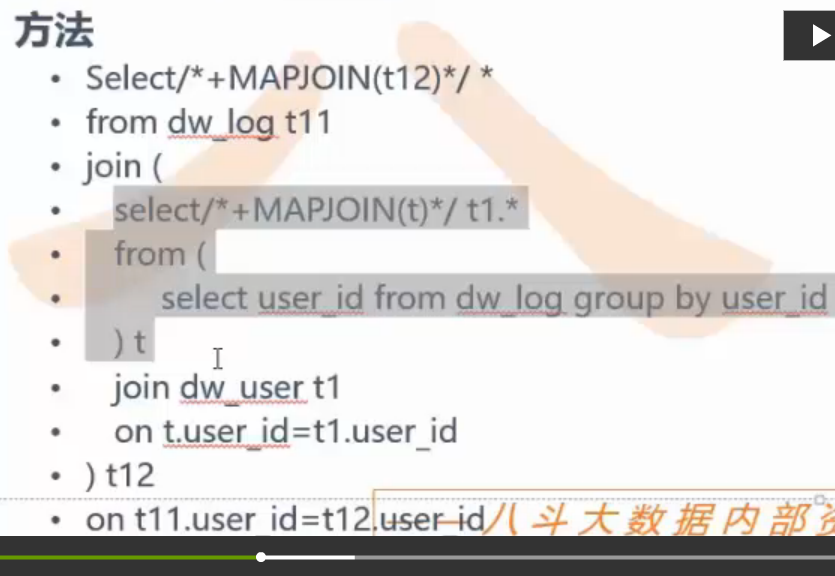
图8：  


这上图8里面的案例也是做了一个join，你会发现一旦join，就会很容易引起需要优化的问题对不对？所以我们的优化大部分是集中在怎么来去完成两个表的join，好了那这个上图8中的表（dw\_log）就类似于一个log对不对？你就类似于一个点击的日志对吧？然后后面上图8中的这个表就相当于是用户的基础属性（dw\_user），然后我对这两张表相同的key去做一个join，那么这个时候比如说这两个表特别特别大，但是你通过上图8中的sql（select \* from dw\_log.t join dw\_user t1 on t\_user\_id=t1.user\_id）语句根本跑不出来，怎办呢？你看看能不能想办法从数据本身的一些特点入手，那你就考虑一下这个日志它是很大，但是毕竟每天登录的用户其实是很少的。

就是说你一天的用户量和一个月的用户（去重的）是完全不一样的，所以你把时间拉长之后，你去做一些数据的统计其实会发现这个用户量其实是很大的，只不过是你从当天这个纬度来看，因为你时间毕竟短嘛，那么你再去统计用户数量的时候肯定是很少的。

那既然我们的需求当天的用户登录很少，其实我们只拿当天的这个记录就可以了，那我们能不能提前对这个数据进行过滤，这就类似于做一个过滤，sql语句：select/\*+MAPJOLN(t)\*/t1.\* from(select user\_id from dw\_log group by user\_id)（ 如下图9所示）

图9：

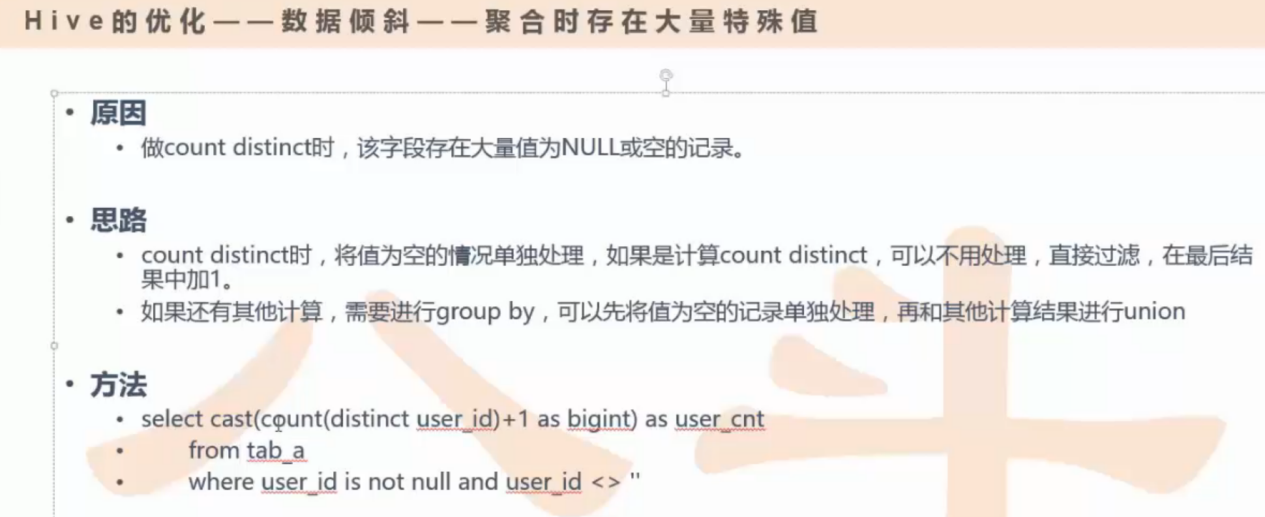


我们来看看它是怎么做过滤的，它相当于是把这个t这个表作为一个小表，然后拿这个小表去和我这个外面这个用户基础属性的这个大表（dw\_user t1）做一个关联，一天的用户，去重的量是很小的对吧？那么单独看这个sql语句select user\_id from dw\_log group by user\_id是什么意思呢？虽然我这个日志表（ dw\_log ）是很大的，但是我由于做了一个group by，那么我最后select出来的这个user\_id的个数是不是很小？那所以如果你把这个user\_id结果当作一个表的话，其实这个id的数量是很小很小的，那么你完全可以把它放到内存里面去，那t这个表通过select/\*+MAPJOLN(t)\*/这样的方式就告诉说你t这个表这个数据是可以放内存的。

那你既然放内存，那就类似于做了一个白名单对不对？也就是说你这个用户基础表（dw\_user t1）它存了所有用户的记录信息，那其实绝大部分用户的基础信息在我这个需求里面根本就用不上，那对于我来说哪些用户基础信息是有用的呢？那么就是当天登录的用户对我来说是有用的，那首先我得把当天的这个用户取出来，因为当天这个用户这个量级是很小的，那你完全可以通过这样sql语句select/\*+MAPJOLN(t)\*/t1.\* from(select user\_id from dw\_log group by user\_id)的方式去完成一个大表和小表之间的一个join得到一个结果出来，然后得到结果出来之后，其实这个数据就已经变得非常小了，这个结果是什么结果呢？相当于是对这个用户信息基础表（dw\_user t1）做了一个过滤，然后上图9里面的join括号里面的这一部分相当于是用户基础表的一个精简，然后精简之后的话你这个时候再和外面这个日志表（dw\_log）做一个交集的话就容易很多了，这个时候因为你这个表变得很小了，那你完全可以把t12这个表尝试的放到内存，可以通过select/\*+MAPJOIN(t12)\*/这样的方式来指定t12可以放到内存里面去，所以这又完成了一次大表和一个小表的一个join，所以相当于这个例子是完成了两次大表和小表的join，并且要结合相应的业务的一些数据的特点完成整个的数据优化。

如果说在你聚合的过程中里面存在了大量的特殊值（如下图10所示）

图10：

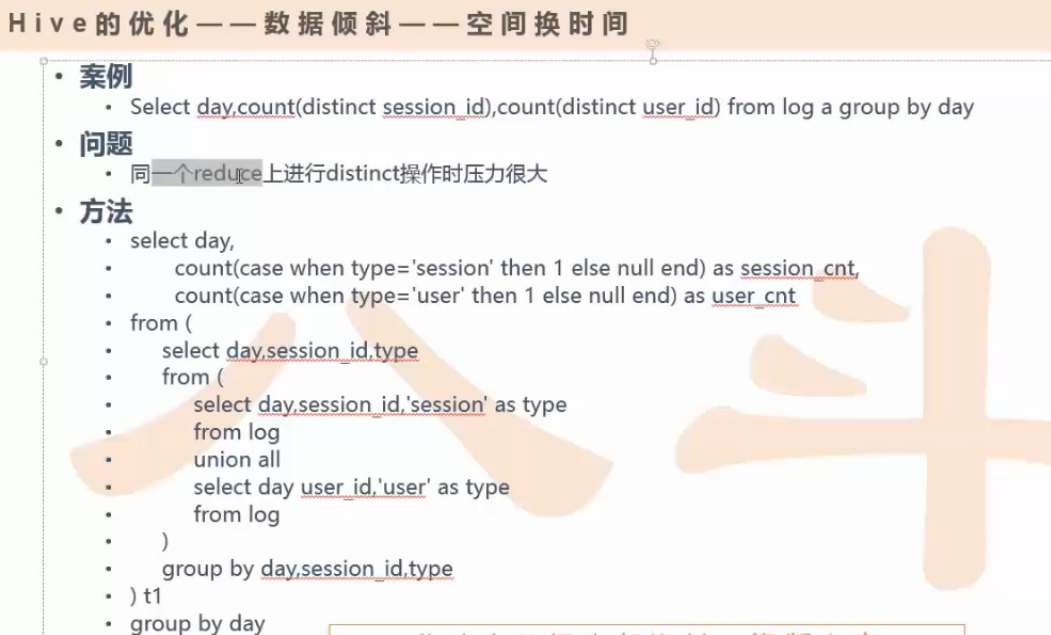


那么这个sql语句select cast(count(distinct(user\_id)+1 as bigint) as user\_cnt是什么意思呢？这里面就讲了一个distinct对吧？这个distinct相当于对userid做一个去重，去重完之后求了个count，我先对userid做去重然后求count，这是一个求去重之后的计数，求userid的个数，如果说这里面存在了一些特殊值，比如说你这个userid存在大量等于空的情况，那能不能把这些特殊值提前的过滤掉，因为等于空的这些值的计数值的量还是比较大的，那如果你把这个量很大的这些user\_id你再参与到这个distinct里面来的话势必会有很大的开销，那提前我把这个null这些空我先给它过滤掉，用where的方式给它过滤掉，过滤完了之后我在这个真实的结果(count(distinct(user\_id)上面加一个1，相当于这个1就把这个null的情况我就给它包含进来了，所以这是一个非常巧妙的一个方法，先把那些不符合规范的那些账户先清理掉，然后把这些不规范的这些账户统一当作一个1，然后直接加(count(distinct(user\_id)这里面进来就等于了上图10所写的(count(distinct(user\_id)+1这么一个情况。

case是一个函数，这个函数是把你里面的返回的值（(count(distinct(user\_id)+1）给它改成一个数据类型，改成一个bigint，它就是转换类型用的

还有一个例子（如下图11所示）

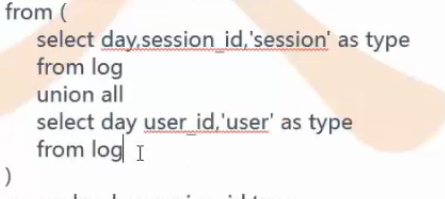
图11：



同一个reduce上进行distinct操作时压力很大，因为这里面做了一些个distinct，因为所有的数据都会到了同一个节点上去了是吧？所以一个reduce压力特别大，那能不能做一个优化呢？那么我们看一下上图11中这里面的例子（select dat.count(distinct session\_id,count(distinct user\_id)from log a group by day）,这个例子就是说这个sql语句的目的是什么呢？就是从这个log日志表里面去读数据，然后以时间（day）为度去做聚合，然后看每天它的session的个数，然后并且它访问用户的一个个数。

就是从这个log里面会把这个session这个计数先捞出来，然后union all一个user，然后把这两个信息进行一个union all，也就是这两个数据是不去做去重的，直接定义的，所以这里面的性能是非常快的（如下图12所示）

图12：



然后当作一个表，然后被上面进行进一步的迭代（如下图13所示）

图13：



然后得到一个day，session和type三个字段，然后下面也是day，session和type进行一个group by（如下图14所示）

图14：



它相当于对这个type做一个汇总，其实这个type它是有取值的，而这个type它里面有两种取值，一种是可以对sesiion取值，一种是对user进行取值，然后它聚合完之后然后求一个count，这个时候当type=session，把这一部分的数据调出来，然后这个时候会做一个count，然后select出来就可以了，这个就是思路（如下图15所示）

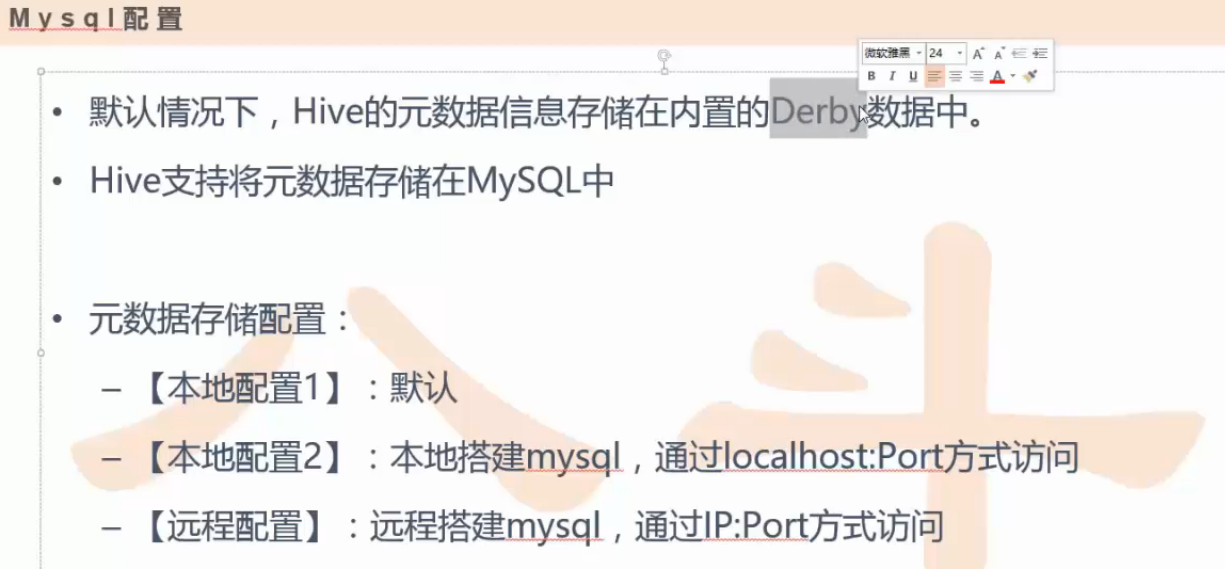
图15：



所以通过这样的形式就避免了通过之前用distinct这种把所有的数据都汇总到同一个reduce上这么一个问题，这也是一个优化。

我们知道之前我们之前讲了一个hive里面不是有一个元数据对不对？什么是元数据呢？元数据就是代表你的数据的数据，存这个数据的表的数据，那么通常叫他元数据会默认的情况下会存在你的一个叫Derby的这么一个数据里面（如下图16所示）

图16：



本地数据库是一个单用户的模式对不对？还有多用户模式和远程服务模式，单用户模式通常是你本地自己玩一玩自己调试一下的，然后多用户模式相当于是你比如在一个节点上多建几个账户是不是？比如说在一个部门内部然后大家共享了一套服务对吧？但是如果你把这个任务抛到整个网络的体系里面去的话，相当于是每一个人都可以做一个节点，然后向远程服务的进行一个网络请求对吧？那么这个时候就建议大家把这个元数据存入你的mysql里面去，所以对于hive的配置的话有三种，一个是默认的本地配置，就是走了你这个Derby这么一个本地数据库。

还有一个本地模式，这个本地模式就是对接着我刚才说过的多用户模式，然后通过一个localhost后面加一个port的方式去访问。

远程模式的话这个就会比较通用了，你在远程搭建一个mysql，通过一个ip的方式去访问。

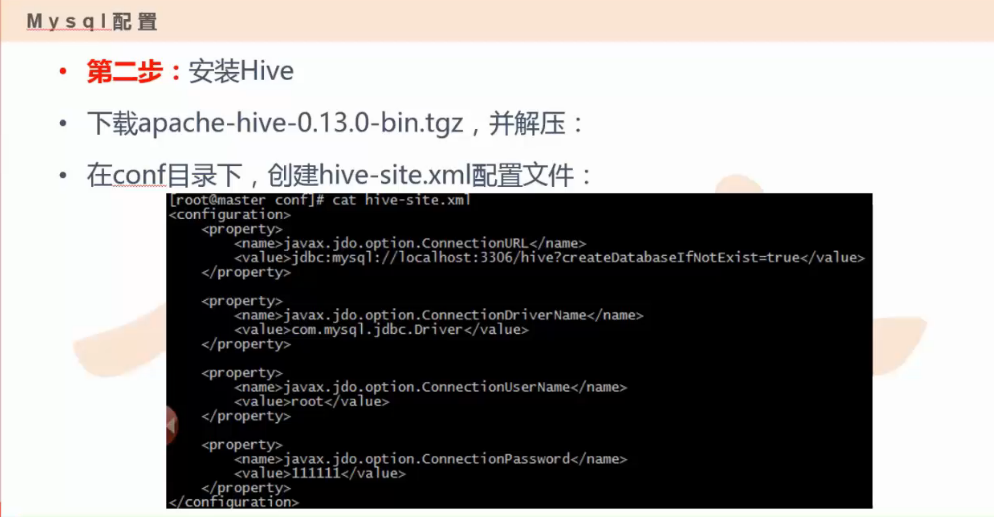
所以接下来我们准备先配置一个把元数据放到我们mysql里面去，那首先你要想把这个hive能够用起来，前提你要先把mysql先跑起来，mysql怎么跑呢？可以根据下图17去做

图17：

那么这里mysql的安装我在这里就不进行演示了，可以根据我上面提供的图17里面的步骤去做即可！（ps:只需要安装在主机器上，其他节点上不用）

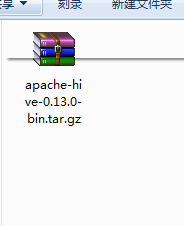
那么接下来我们看一下如何安装hive（如下图18所示）

图18：



把hive的包放入我们的集群上先（如下图19所示）然后进行解压

图19：



然后因为这个hive并不是一个集群，它只是一个工具，所以你在集群上解压完了这个hive工具包以后，就不需要同步到其他节点上去。

然后就开始改配置，就创建一个hive-site.xml配置文件，然后把下图20和21配置添加上去即可

图20：

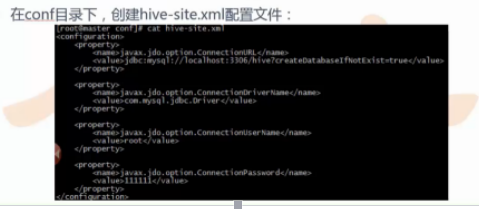
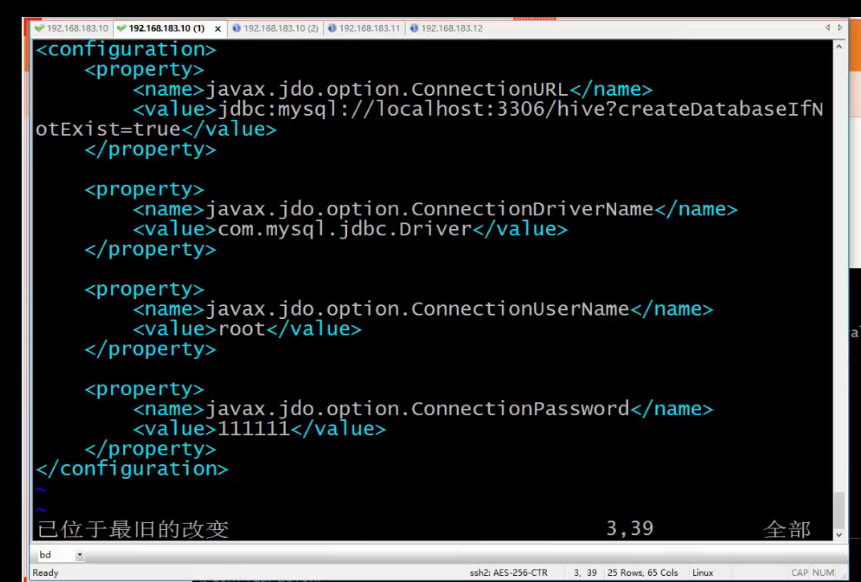
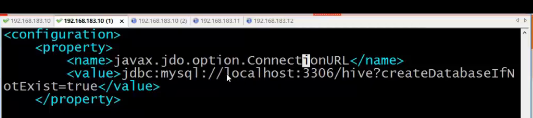


图21：



然后首先第一个就是元数据存在的地址，我们是通过mysql，因为这个mysql是在本地安装的，所以你可以通过localhost，如果你是远程安装的，那么你就可以把localhost改为ip。所以我们通过这个配置就可以看出来我们现在搭建的是本地配置2（如下图22）

图22：



还有一个你需要配置mysql里面有一个接口驱动，还要把一个驱动复制到hive的目录的lib下里面（如下图23和下图24和下图25和下图26所示）

图23：

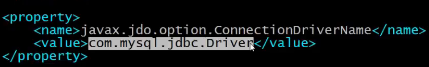


图24：  


图25：

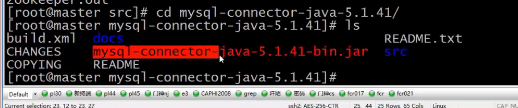
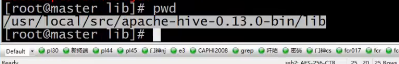
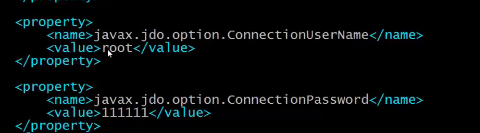


图25：



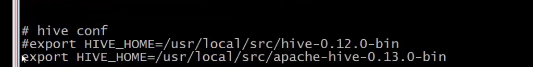
然后接下来就可以访问mysql了，那这个时候就要指定你的用户名和密码（如下图26）

图26：



然后这个时候你就把hive安装的差不多了，接下来你要在~/.bashrc里面把这个hive解压后的目录放进来（如下图27所示）

图27：



然后放入再到path目录下就可以了（如下图28所示）

图28：



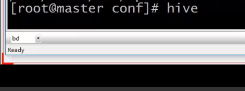
然后这些配置配好以后，就在命令行输入bash，那么相当于这个hive就可以提供支持了（如下图29所示）

图29：



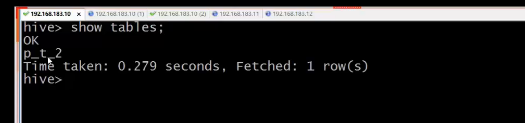
然后再执行hive就可以启动了（如下图30所示）

图30：



然后进行测试，看看hive能不能用（如下图31所示）

图31：



实践视频笔记会在后续进行一个更新！敬请期待！