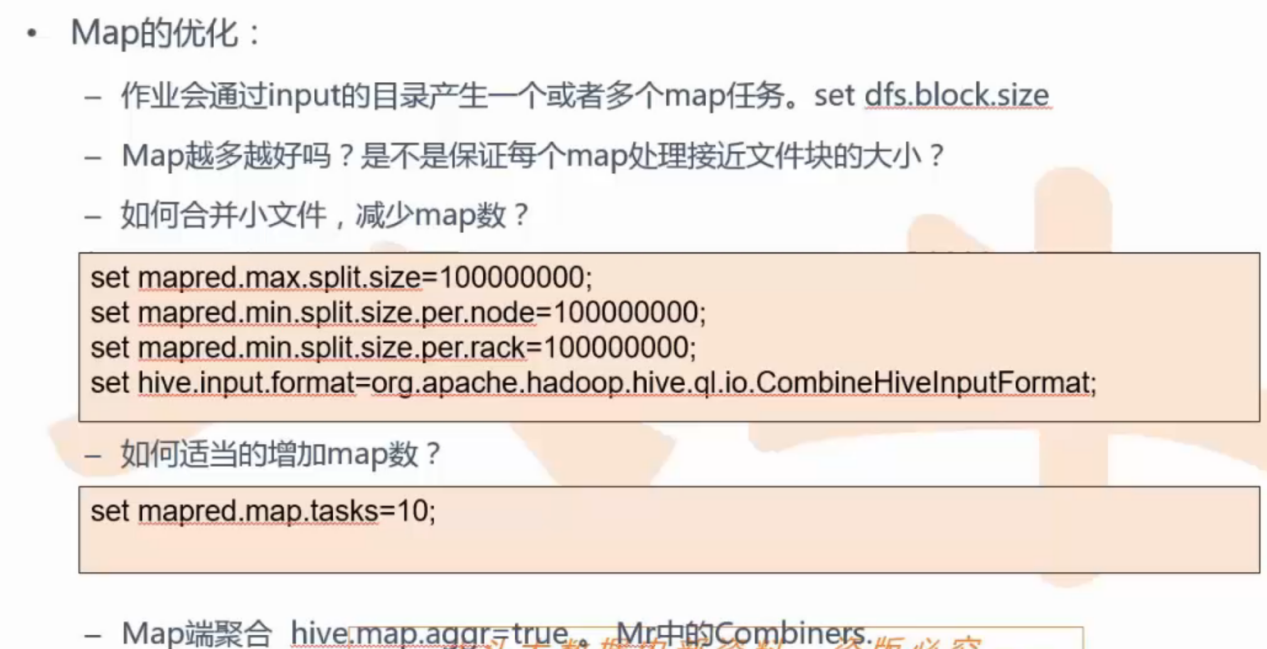
还有一些扩展的内容，就是属于一些大家在以后使用hive的过程中接触到的一些优化，那么我们来看一下hive的优化部分（如下图1所示）

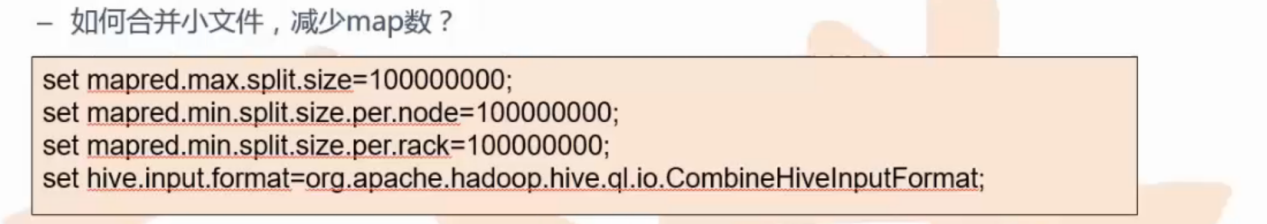
图1：



首先hive的优化跟mapreduce的优化有一些类似，就是说我们在mapreduce里面那如果一个任务它跑得快还是慢，跟它设置的map的个数以及reduce的个数有着很大的关系，那有一些参数的话如果你设置的合适的话那就会给你整个的任务带来一些很好的效果，但是说不合适的话反而变得很差对吧？那这里面其实一样，因为hive它本质上是mapreduce，所以个别的情况也会考虑到，为了让增加并发，怎么来调整并发度，我们之前讲的mapreduce的时候有这么几种方法对不对？那第一个想到的方法可以调整它的blocksize对吧？因为每一个数据它的最基本单位是一个block对吧？那你block的大小直接影响到了你去做map切分的这些sqlit的份数，然后这每一个sqlit对接一个map，如果你切分的更细一点的话，那你sqlit的个数就会增加，那直接会影响到你的map个数也会增加，所以blocksize是一个直接影响到你的并发度的这么一个很关键的因素，HDFS你是怎么设blocksize的呢？有一种方法我们可以去通过HDFS有一个size.xml上面有一个配置可以改，但是这个改的话，相当于是对全局的blocksize做修改了，整个公司可能就你这个任务受到了一些blocksize的一些大小不合适的影响，难道你就为了你这个任务去调整全局的一个block的一个size然后影响到其他所有的这个任务吗？那么这个肯定是划不来的对不对？所以怎么办呢？

那我们只能从我们自己提交任务的脚本，看看能不能通过一些配置去进行设置，就类似于下面一些配置（如下图2所示）

图2：



其实这个也类似在我们讲mapreduce的过程中也类似提到了这样的一个类似的方法对不对？你可以通过改这些参数然后调整它的map的个数，另外还有一种方法就是你可以设置一个这样的方式（如下图3所示）

图3：



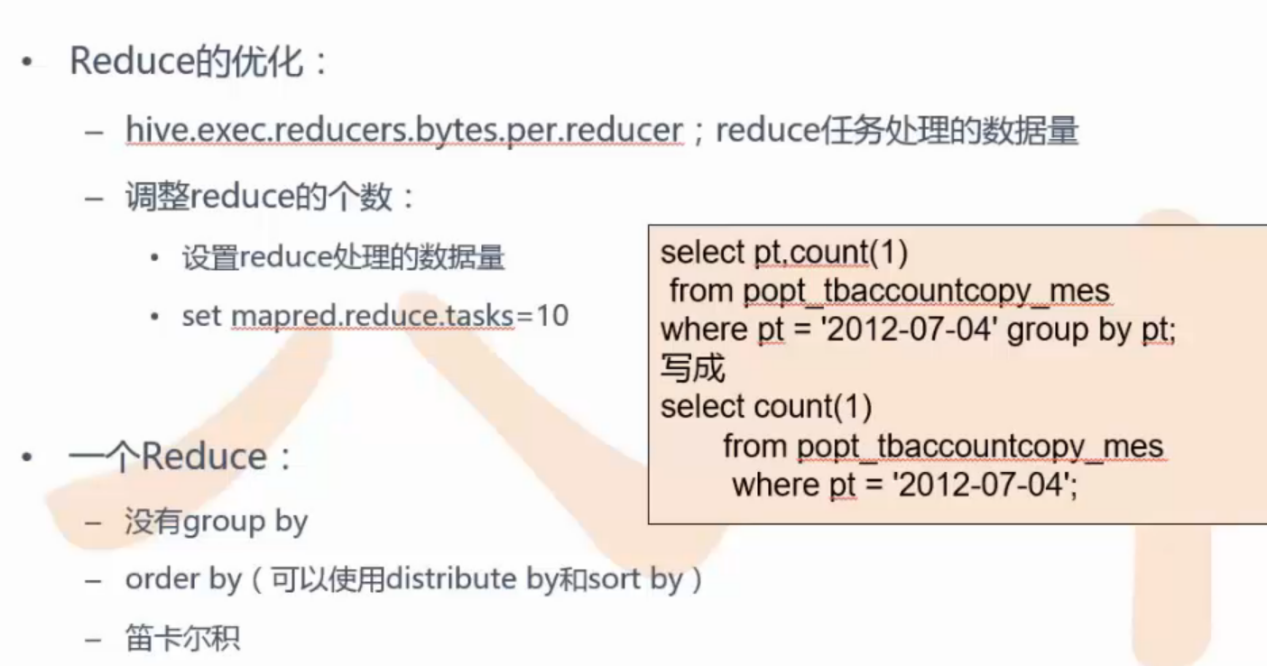
管好mapreduce你可以设置一个map的个数，直接可以设一个个数，但是我们之前说过了，这个个数不一定生效对吧？它只是一个参考却不一定生效，还有一个为了让我们的任务能干加快有一些任务你可以考虑开进你的Combiner，这个功能是可以直接变成true的（如下图4所示）

图4：



然后对应着map的优化呢，reduce的个数也是会经常做一些个对应的调整（如下图5所示）

图5：



这个hive.exec.reducers.bytes.per.reducer这个参数是可以你提交sql的时候然后去设置这么一个参数，这个数字是会直接影响你的reduce的一个个数，这个reduce的个数为什么可以受到这个hive.exec.reducers.bytes.per.reducer参数的影响呢？是因为这里面是有一个公式，那么这个公式大家记一下，有一个reduce的计算公式，但是这个计算公式是依赖于你这个hive.exec.reducers.bytes.per.reducer参数的。

reduce的个数等于有两部分组成，一个是InputFileSize除以bytes字节per reducer

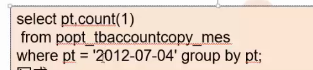
reduce的个数= InputFileSize / bytes per reducer，大家看这个公式应该能看懂对吧？我整体的文件的大小，整个数据的大小除以每一个reduce它可以处理的大小是不是？那总大小处理每一个节点处理的大小那就是相当于是它一共可以采用多少个reduce节点对吧？那么这个参数hive.exec.reducers.bytes.per.reducer设的值是哪里的呢？就是通过这个公式（reduce的个数= InputFileSize / bytes per reducer）可以计算得到一个reduce的一个个数，可以通过这个方法去设置，但是这个reduce的个数的话不一定是按照你要求那样就是说我就想要10个reduce，那么这个就不一定能够保证对不对？

那么怎么样能够确定你这个reduce你想要的那个个数呢？比如说我就要reduce个数为100个，那么这个是通过另外一种方式就是通过上图5中的参数(set mapred.reduce.tasks=10)，

这个参数set mapred.reduce.tasks=10大家很熟悉啊，我们在讲mapreduce里面不是设了一个reduce的个数吗？那它的参数跟这个set mapred.reduce.tasks=10是一样的，你直接set这个参数等于10，那么它最后跑reduce个数的时候就是10个reduce，用10个reduce去帮你并发，那如果说这两个参数hive.exec.reducers.bytes.per.reducer和set mapred.reduce.tasks=10同时设置的话，那么就以set mapred.reduce.tasks=10这个参数为准。

然后后面这里还有一个例子（如下图6所示）

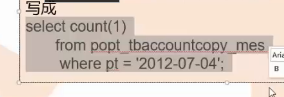
图6：



这个sql里面涉及了一个group by，group by什么意思大家想必也知道了把？group by后面的某一个字段然后进行聚合，那相当于它从某一个表（popt\_toaccountcopy\_mes）里面去读一个数据，对这个数据把这个pt字段进行输出，并且按照这个字段出现的一些个数然后进行一些个计数count对不对？然后做一些个条件判断，所以这个上面图6的例子是比较简单的例子。

那么如果你没有按照上图6那样的方式去写，我写成这个样子会有什么问题？（如下图7所示）

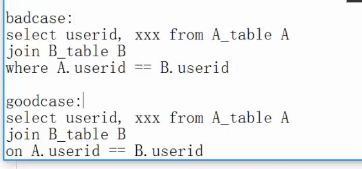
图7：



虽然这个它最后的功能也实现一个计数，但是最后呢你跑完了这上图7的sql，它对应的mapreduce它就用一个reduce来运行，所以对于一个没有group by，而你有做了count这种聚合类的函数的时候它最后跑出的reduce的个数肯定是一个redeuce，特别是面对这种大量的数据，如果你没有做group by，那么直接如果按照全局去count的话，那它相当于没有做一些优化，那建议大家可以考虑相应的group by做一定的聚合，这样的话能够增加一些个并发。

我们通常写sql的时候比如说select userid xxx from A\_table A，join B\_table B 这样就代表用后面这个缩写的A和B然后代替前面的table名字，然后如果说我想对这两个表做join的话，那where A.userid == B.userid的话，相当于是我对同一个key去做一个聚合，然后把这两个聚合起来然后我想输出一些相应的数据（如下图8所示）

图8：



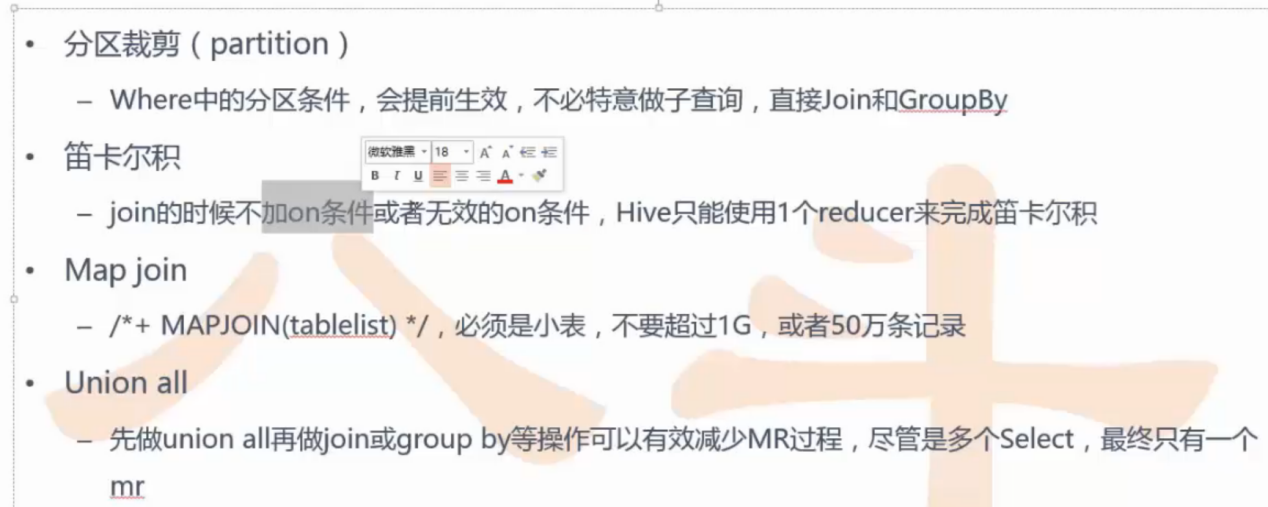
就是建议大家，join的时候如果你要对两个表聚合的时候，如果是不用这种带on的情况下它最后也会一个reduce来运行。

那么相当于上面那个就是一个badcase，下面的是goodcase。

一般这种两个表做join，那肯定条件会放到on里面对不对？

好了，然后呢我们还有一个优化，这是我们之前讲过的，有一个partition这个比较好理解，就是通过不同的目录然后命中不同的目录，把那些条件然后尽量的可以定位相应的分区来去减少一些不必要的数据的过滤对不对？（如下图9所示）

图9：



然后笛卡尔积其实也说过了，如果是join的时候不加on条件或者是加一个无效的on条件，那么这个时候hive只会用一个reduce来完成笛卡尔积。

还有Map join这个比较重要，这个有点像什么呢？这个有点像查找配置这么一个过程，像去map join类似一种检索配置文件的形式，就是说我有两个表，一个A表一个B表，如果A想join这个B，那默认情况下相当于是这个A是一个文件，B是一个文件，然后之前我们做mapreduce相当于导入两个文件，然后在硬盘上会做一些个交换，因为mapreduce里面会经常和你的磁盘进行交互对吧？那假设说如果有一个A或者一个B，其实有一个表它是一个小表，比如说A是小表，那么考虑是否可以将A放入内存这个大家明白了把？如果这个A这个表它可以提前放入到内存的时候，那相当于你就对这个B进行一个遍历就可以了，遍历的过程中然后进行这些A的这些查询，就相当于在内存中完成一个数据的一个连接，刚才说了我假设A是小表，如果B是一个大表，那相当于我可以对这个B进行一个从头到尾的一个遍历，然后A因为它是小表所以它可以放到内存里面，然后遍历B的过程中那可以对相应的字段进行查内存，把相应的数据直接输出，然后给它拼上这么一个过程。

小表多小？一般来说不要超过一个G就基本上比较小了，这个要看你的内存，但是尽量不要超过一个G，那怎么来认为哪个表是小表呢?比如我这里写sql的时候，这个tableA和这个tableB我根本看不出来哪个是大表哪个是小表，那么其实你可以通过一个参数（/\*+MAPJOLN(tablelist)\*/）这样的形式，然后来标识哪个表是小表明白了把？可以通过这样的方式去明确一个，就是可以把这个table明确这个是大表还是小表。

还有一种就是union all，这个union操作想必大家还不知道什么意思把?union操作类似于是一个合并去重操作，比如说你有一个ABC，然后这是一个T1表里面有ABC，然后还有另外一张表叫做T2，然后T2里面有BCD，然后我对这两个表做一个合并，合并之后的的数据是ABCD是不是？这就是union的结果。

那么对于一个union all的结果是什么呢？就是一个直接追加的数据，相当于是把两个表的数据放在了一块，得到的数据是ABCBCD，它就不做去重，所以union是做去重的，而union all的是不做去重的，是直接追加数据的。

所以你会发现union all的缺点就相当于是它没有去重对吧？然后这个数据量直接叠加，感觉非常粗暴对吧？但是它的优点就是说它的操作非常简单是吧？没有太多的性能消耗是吧？然后速度就是快。所以union all的性能要明显的高于union ，所以如果是你当确认有多个记录集不存在有相同的记录，或者是有可能有相同的记录但是明确可以要求合并的，可以用union，即使有相同的记录也不需要合并的话就用union all，所以大家了解一下union all和union 的一个区别。

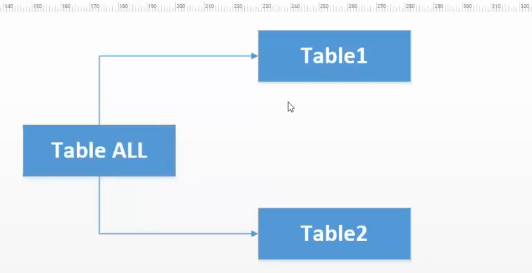
然后我们再看一下，如果我要是有两个不同的任务，那它的数据源都是来自于一个表的话（如下图10所示）

图10：



比如类似于这个例子（FROM from\_statement），我想从这个公共的表里面取出数据，然后在去写到另外一个表里面去，就相当于是（如下图11）这个样子

图11：

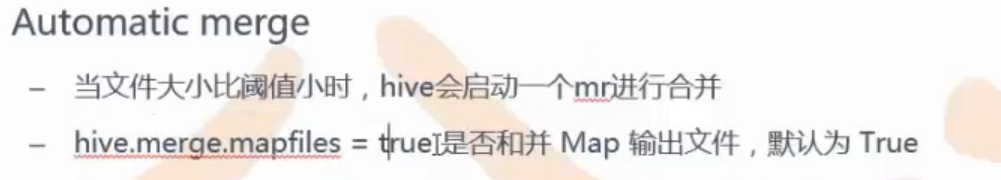


就是说Table1的内容是来自于Table，Table1相当于是Table all的一个子集，然后Table2也是一个Table all的子集对不对？那这个时候可以把这个语句直接写成这个样子（如下图12所示）



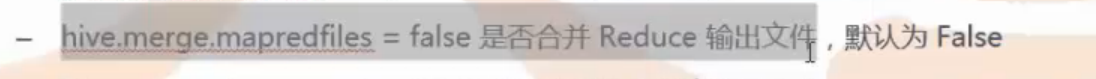
直接FROM from\_statement一个总表然后进行两次子表的一个生成，还有在整个的hive过程中你执行一个hive的任务产生了一些临时的中间文件对不对？然后小的文件会比较多，那怎么样让多个小文件进行合并呢？那其实你也可以配置一些参数，那这些参数就类似于我们之前讲Mapreduce一样，你可以在中间做一些文件的一些合并（如下图13所示）

图13：



就是在map阶段进行合并，这个是提交任务的时候直接去配的，然后这个是在reduce的结果做一些个合并（如下图14所示）

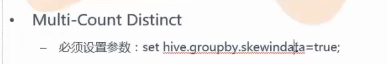
图14：



这个是在reduce的结果上做一个合并，然后可以设置一些每一个任务它的预期合并的这个文件的size大小可以通过这样的一系列配置然后去调整它整个的小文件的一个产生的一个控制情况，这也是一个优化的地方。

还有一个就是很有用的一个优化，但是你想开启这个优化的话你必须要设置一个参数，你把这个参数设置一个true（如下图15所示）

图15：



那把它设置一个true有什么用呢？那如果有一个sql，这个sql里面会有什么呢？会有一个group by key 它这个功能你会发现它会对某一个key做一个聚合是吧？然后它会生成比如一个mapreduce，那么这个mapreduce怎么写呢？相当于是把map阶段相同的key，因为你是key去做聚合的，那相同的key最后会传输到同一个reduce里面去进行处理。

所以如果假设说如果这个数据源存在数据倾斜问题，就是说有的key的记录多，有的key的记录少是不是？如果那些key的记录多的那些数据会跑到同一个reduce里面去，这个时候那跑到同一个reduce里面，那它这个记录就处理的时间会比较长是不是？，个别的key它的聚合记录少，那相应那个reduce里面处理的记录就不会那么多，相当于是比较空闲一点是不是？那这个导致比如说你最后有多个reduce的情况，比如说有10个redice来跑，那最后可能大家都等一个reduce来完成，所以这就是数据倾斜的问题，就是key的分布不均匀，因为你按照key进行group by，那所以同样的key的记录都会发送到同一个reduce里面去。那既然相同的key的话放在同一个桶里面去，那肯定会存在有的桶多有的桶少，比如说有一个key它背后的记录特别特别大，但是没有办法，它必须要全部塞到同一个桶里面去，那么被接收这个桶它要处理的数据是非常非常大的，所以就有可能存在大多数的reduce都已经完成了，结果就还剩下一个reducde，相当于这是一个数据倾斜的一个严重的表现。

我们这里面就有一个优化，我们在写mapreduce的时候，其实对于这种情况是可以做优化的，我们可以对这个key进行一个修改，相当于把以key为group by的那些记录比较多的呢看看能不能把它拆开，就是做一个对key进行一个改写来解决这个问题，这是我们以前写mapreduce的一个经验。

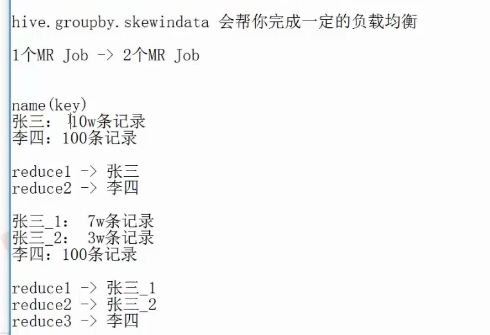
但是我们现在已经开始写hive了，我们写hive的话，肯定是不会记录太多的人工的一些个修饰，那么这里面就有一个很重要的参数就是（set hive,groupby.skewindata=true），你把这个参数设置了之后，那它就会自动的帮你完成一个负载均衡，怎么做负载均衡呢？我们看看能不能把一个mapreduce job变成两个mapreduce job。

那这里我举个一个通俗易懂的例子，假设有一个张三，这个人后面大概有10万条记录，李四这个人背后可能就有100条记录，所以这个数据一看就是差距非常非常大，那比如说redecue1来处理张三的问题，然后reduce2来处理李四的问题，这时候你会发现它有一个严重的数据倾斜，那么我们来看看能不能把张三这个人变一下，变成多个，变成一个张三\_1，张三\_2，然后张三\_1变成了7万，张三\_2变成了3万，相当于把原来这个大量的数据可以拆分，拆分出多份，然后尽量保证后面每一个reduce它处理的数据量是基本能够均匀的。

但是你会发现，即使这种情况的话到最后得到的结果是不对的，其实我可以把张三\_1和张三\_2和李四这三个的输出做一个临时结果，然后再放入另外一个maoreduce里面去，所以这个时候第一个redcuce相当于把我的数据规模严重的压缩，然后第二个reduce完成一个最终的一个聚合，这个类似于一个Combiner的思想，但是还不太一样，就是说它第一个阶段，它分两个阶段，第一个阶段它虽然是相同的key但是我可以允许同样的key的记录可以分发到不同的reduce上面去，我可以先有一个忍耐度对吧？我可以对同样的key对后面不同的记录可以分发到不同的节点上去，但是最后一个reduce的话，你必须要把前面的那个记录全局的汇到一个reduce里面去。

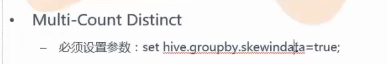
所以你如果直接一步到位，通过一个mapreduce来跑的话肯定会有一个数据倾斜的一个问题，那如果把这个问题拆开，拆为两个部分，那这个问题就可以有效的进行一些个减缓，最终的目的是为了保证一个负载均衡，就是说我第一个reduce，我就考虑后面每一个节点的一个数据的一个均匀程度，但它不能保证相同的key的记录能够跑到一个reduce里面去，这是第一个mapreduce。

第二个mapreduce因为前面这个处理完了之后，中间数据规模就已经远远小于它的数据源的规模了对不对？，所以接下来的操作的话就比较灵活了（如下图16所示）

图16：  


所以你就可以通过这个group by key的语句就可以看出，这个变量是能够控制一些负载均衡的，当出现了一些数据倾斜的一些现象的时候，那如果你这个参数设置为这个true，那么hive会自动的帮你做一个负载均衡，这是一个优化。（如下图17所示）

图17：



好了还有一个，就是通常来说你要跑一个mapreduce做一个什么工作会比较耗时呢？肯定是reduce会比较耗时，我这里有两个sql语句（如下图18所示）

图18：

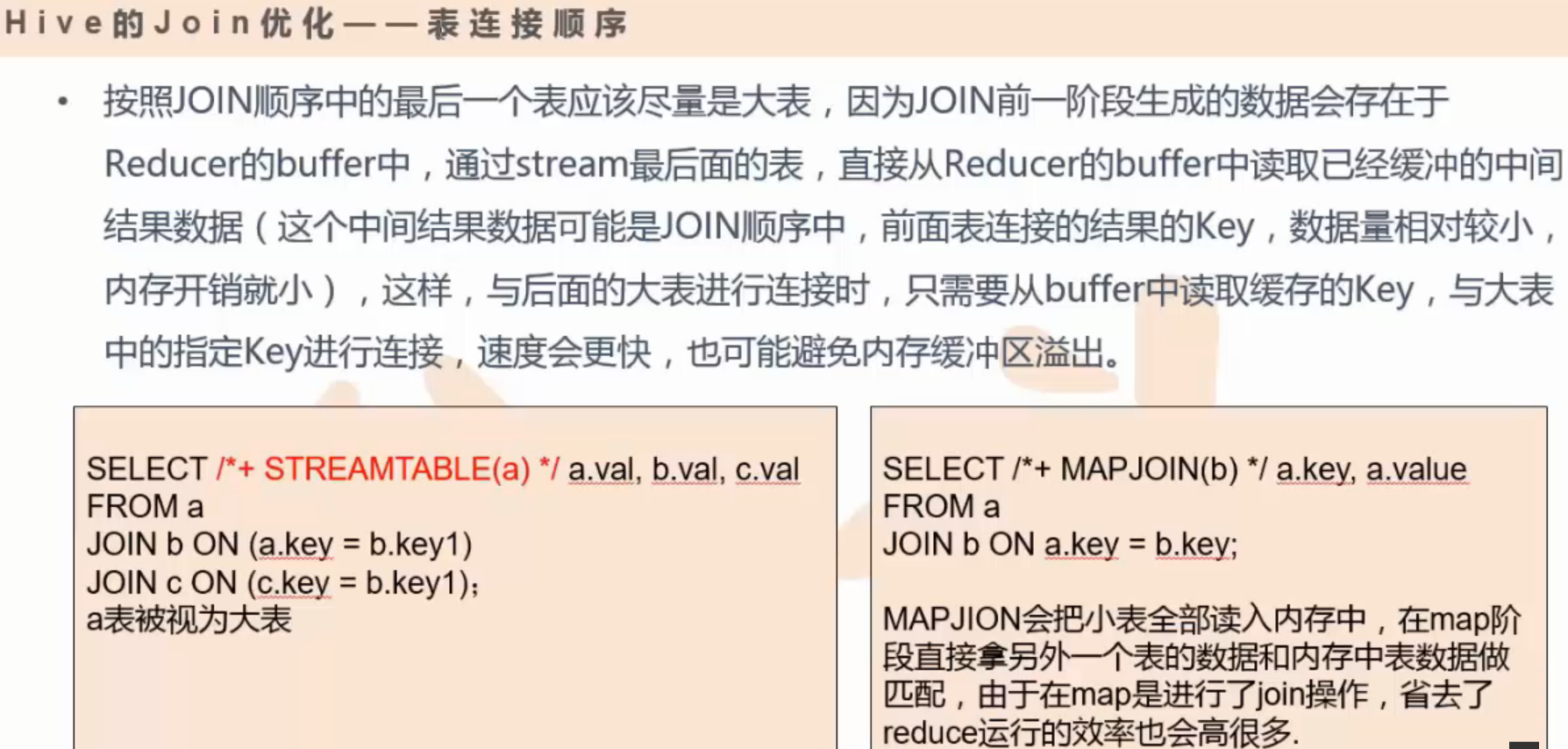


这两个sql语句有什么区别呢？唯一的区别就是这个条件不一样，那么上面这个一个MRjob里面相当于是左边的key都是一个key，都是来自同一个表，下面的多个mpjob就是来自于一个不同的表对不对？，但是这两种写法的差异就很大了，那么上面那种写法只能生成一个mapreduce，而下面那个那种写法相当于是生成多个maprediuce。

我们的优化目标是让它mapreduce越少越好对不对？因为通常来看reduce越多的话会耗时的时间会更长，所以上图18中的生成一个mapreduce是一个优化的方法。

你在两个表做join的时候，如果一个表是小表的话，你可以把这个表提前放到内存里面去，然后过另外一个后面的大表的时候，它会帮你做一个性能的提升，这里面就会考虑到了一个问题，就是一个表的连接顺序的问题（如下图19所示）

图19：



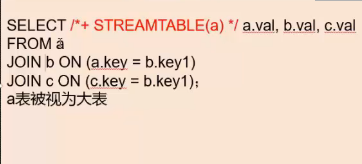
就是说你在做join的时候，每次mapreduce的任务逻辑是这样的，就是reduce会缓存join序列中除了最后一个表的所有表记录，就是比如说A.join(B)，如果一旦涉及到join的话，那通常会引发一个reduce对不对？那既然引发reduce的话，那在它的内部机制的实现是这样的，就是说我这里有两个表，一个是表A，一个是表B，那我除了最后一个表的时候，那么这个reduce会尝试A表进行缓存，什么是缓存呢？就是它倾向于把A这个表进行往内存里面去写是吧？如果是A表joinB表的时候，因为A在前面，B相当于是整个语句里面后面一张表对不对？那整个的变成一个mapreduce过程中，它默认会除了B 以外的前面的所有的表，都把这些表尝试放入缓存中，这个是它的默认情况，虽然我没有指定A这个表是小表，但是它会尝试的把这个表当作小表进行缓存，这个是一个很好的一个优点，就是说如果A完全可以放到内存里面去，那这个时候我可以对B进行一个遍历，然后进行join。

如果这个B长度是n，如果这个B它有n条记录，那它的时间复杂度上就是O（n），所以你把其中一个表当作缓存提前放到内存的话，那它会时间复杂度优化是一个很好的优化。

比如说写一个select x ,y from A join B，那么我们在去写这个语句的时候，就可以提前判断一下哪个表是小表，哪个表是大表，如果是哪个表是小表的话，尽量的把这个表放到这个join的前面，所以对于一个刚开始写sql的时候，他可能对这个细节不是特别的了解，但是如果要是对一个有经验的一个写sql语句的人来说，他就会自然有意识的会把小表放到join的前面，因为hive内部的机制会对这个做一定的优化。

所以hive会默认把左边的数据放到内存，右边的数据做类似的流数据，因为右边这个B他这个数据记录他是从头到尾进行一个遍历，好了但是这种方式的话，也不是特别的好，假设说我这个B和这个A我就写反了，B写到join的前面了，A写后面了，那这样的话会对它内部的优化机制就不一定派的上用场了，那这个时候，我不想改这两个之间的顺序，但是我还想利用它内部的一些优化机制，这个怎么做呢？

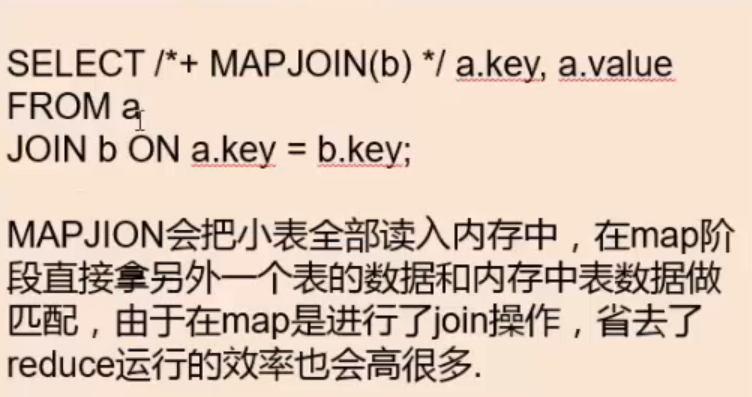
就可以考虑这种方式（如下图20所示）

图20：  


就是通过这样/\*+STREAMTABLE(a)\*/参数，我来指明哪个表是大表，所以这个大表我们刚才说了，就是右边这个表类似于做流数据，既然做流数据，那么就是stream对吧？，那么你可以看看这个参数叫STREAMTABLE，用来做流数据的表，那么这一个表是一个非常大的表，那这个时候它会明确指定a表是一个大表，虽然在这个a表在join的前面，但是我可以指明a表是大表，然后它就可以把b和c尝试做一个小表进行缓存。

那么还有一个例子也是有类似的（如下图21所示）

图21：



但是这个上图21的图里面类似的我们刚刚说过对吧？你可以用map join的方式来指明哪个表是小表，这上图21里面是可以通过 /\*+MAPJOIN(b)来指明谁是小表，那么上图21中b是小表，那么虽然我a排在了Join的前面，讲道理来说它会系统把a自动的当作一个小表来处理是不是？然后b作为一个大表，但其实a这个表的记录要远远大于b，如果你a表放到内存，那你对b其实遍历也遍历完了是吧？相当于你没有把这个优化做到很到位，那你可以通过这个/\*+MAPJOIN(b)参数的方式来指明b这个表是一个小表，然后它就会把b放到内存里面去。

所以可以通过这/\*+STREAMTABLE(a)\*/和 /\*+MAPJOIN(b)两个参数来去控制你join前后这个表的一个大小的一个情况。所以最好是每次写join的时候，小表放左边，大表放右边

例如对于几十万或者上亿的表，如果不做优化和做了优化相比的话，那不做优化的性能消耗要比这个做了优化的性能消耗起码要多出很长时间，那具体多少倍，就要你具体的数据规模，因为像这种sql你写完之后，通常在线上用的时候不是说你写完一次就完事了，经常对于那例行任务的话，那你肯定是要让这个任务达到最优势越好对不对？