**表优化**

1.动态分区（不同文件夹）：  
set hive.exec.dynamic.partition=true;  
set hive.exec.dynamic.partition.mode=nonstrict;

**默认值：strict  
描述：strict是避免全分区字段是动态的，必须有至少一个分区字段是指定有值的**

2.分桶（不同文件）：  
set hive.enforce.bucketing=true;  
set hive.enforce.sorting=true;

**默认值：false  
描述：开启强制排序，插数据到表中会进行强制排序**

1. 队列：

set mapred.queue.name=queue3;设置队列queue3  
set mapred.job.queue.name=queue3;设置使用queue3  
set mapred.job.priority=HIGH；

队列参考文章：  
http://yaoyinjie.blog.51cto.com/3189782/872294

**Job优化**

1. 并行化执行  
   set hive.exec.parallel=true;

**默认值：false  
描述：hive默认job是顺序进行的，一个HQL拆分成多个job，job之间无依赖关系也没有相互影响可以并行执行**set hive.exec.parallel.thread.number=8;

**默认值：  
描述：就是控制对于同一个sql来说同时可以运行的job的最大值，该参数默认为8.此时最大可以同时运行8个job**  
2.本地化执行（在存放数据的节点上执行）  
set hive.exec.mode.local.auto=true;  
**本地化执行必须满足条件：  
（1）job的输入数据大小必须小于参数  
hive.exec.mode.local.auto.inputbytes.max(默认128MB）  
（2）job的map数必须小于参数：  
hive.exec.mode.local.auto.tasks.max(默认为4）太多没有足够的slots  
（3）job的reduce数必须为0或1**  
3.job合并输入小文件  
hive.input.format=org.apache.hadoop.hive.ql.io.CombineHiveInputFormat  
**默认值：false  
描述：多个split合成一个,合并split数由mapred.max.split.size限制的大小决定**  
  
4.job合并输出小文件（为后续job优化做准备）  
set hive.merge.smallfiles.avgsize=256000000;

**默认值：  
描述：当输出文件平均大小小于该值，启动新job合并文件**  
set hive.merge.size.per.task=64000000;  
**默认值：  
描述：合并之后的每个文件大小**5.JVM重利用  
set mapred.job.reuse.jvm.num.tasks=20;

**默认值：  
描述：每个jvm运行多少个task；JVM重利用可以使job长时间保留slot，直到作业结束。**

1. 压缩数据（多个job）  
   （1）中间压缩处理hive查询的多个job之间的数据，对于中间压缩，最好选择一个节省cpu耗时的压缩方式  
   set hive.exec.compress.intermediate=true;  
   sethive.intermediate.compression.codec=org.apache.hadoop.io.compress.SnappyCodec;  
   set hive.intermediate.compression.type=BLOCK;

**默认值：  
描述：按块压缩，而不是记录**  
（2）最终输出压缩（选择压缩效果好的，减少储存空间）   
set hive.exec.compress.output=true;  
setmapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec;  
set mapred.output.compression.type=BLOCK;

**默认值：  
描述：按块压缩，而不是记录**

**Join优化**

hive.optimize.skewjoin=true;

**默认值：false  
描述：如果是join过程出现倾斜，应该设置为true**set hive.skewjoin.key=100000;

**默认值：  
描述：这个是join的键对应的记录条数超过这个值则会进行优化，简单说就是一个job变为两个job执行HQL**

**Group by优化**  
hive.groupby.skewindata=true;  
**默认值：  
描述：如果group by过程出现倾斜应该设置为true**set hive.groupby.mapaggr.checkinterval=100000;

**默认值：  
描述：这个是group的键对应的记录条数超过这个值则会进行优化，也是一个job变为两个job**

count distinct优化  
优化前（只有一个reduce，先去重再count负担比较大）：  
select count(distinct id) from tablename;  
优化后（启动两个job，一个job负责子查询(可以有多个reduce)，另一个job负责count(1)）：  
select count(1) from (select distinct id from tablename) tmp;  
select count(1) from (select id from tablename group by id) tmp;  
set mapred.reduce.tasks=3;

Mapjoin&bucketjoin

1.mapjoin(map端执行join）  
启动方式一：(自动判断）  
set.hive.auto.convert.join=true;  
hive.mapjoin.smalltable.filesize 默认值是25mb  
**描述：小表小于25mb自动启动mapjoin**  
启动方式二：(手动）  
select /\*+mapjoin(A)\*/ f.a,f.b from A t join B f on (f.a=t.a)

mapjoin支持不等值条件  
reducejoin不支持在ON条件中不等值判断

2.bucketjoin(数据访问可以精确到桶级别）  
使用条件：1.两个表以相同方式划分桶  
          2.两个表的桶个数是倍数关系  
例子：  
create table order(cid int,price float) clustered by(cid) into 32 buckets;  
create table customer(id int,first string) clustered by(id) into 32/64 buckets;  
select price from order t join customer s on t.cid=s.id;

where条件优化  
优化前（关系数据库不用考虑会自动优化）：  
select m.cid,u.id from order m join customer u on m.cid =u.id where m.dt='2013-12-12';  
优化后(where条件在map端执行而不是在reduce端执行）：  
select m.cid,u.id from （select \* from order where dt='2013-12-12'） m join customer u on m.cid =u.id;

**Map优化**

1.set mapred.map.tasks=10 无效  
（1）默认map个数  
default\_num=total\_size/block\_size;  
 (2)期望大小（手动设置的个数）  
goal\_num =mapred.map.tasks;  
（3）设置处理的文件大小（根据文件分片大小计算的map个数）  
split\_size=max(block\_size,mapred.min.split.size);  
split\_num=total\_size/split\_size;  
（4）最终计算的map个数（实际map个数）  
compute\_map\_num=min(split\_num,max(default\_num,goal\_num))  
  
总结：  
（1）如果想增加map个数，则设置mapred.map.tasks为一个较大的值；  
（2）如果想减小map个数，则设置mapred.min.split.size为一个较大的值。  
   
2.map端聚合  
set hive.map.aggr=true;相当于map端执行combiner  
  
3.推测执行（默认为true）  
mapred.map.tasks.speculative.execution

**Shuffle优化**  
Map 端  
io.sort.mb  
io.sort.spill.percent  
min.num.spill.for.combine  
io.sort.factor  
io.sort.record.percent  
  
reduce端  
mapred.reduce.parallel.copies  
mapred.reduce.copy.backoff  
io.sort.factor  
mapred.job.shuffle.input.buffer.percent