

//基于规则的逻辑优化  
plan/optimizer.go  
logicaloptimize()

//列裁剪  
plan/rule\_column\_pruning.go  
columnPruner.optimize()

//从逻辑计划的根节点，调用各逻辑计划接口实现的列裁剪  
列裁剪并调用子孩子节点的列裁剪函数  
未实现的函数如**LogicalLimit**，**LogicalSort**等逻辑计划算子  
成员**baseLogicalPlan**，然后递归调用子孩子的列裁剪函数  
//注：虽然函数名和功能叫列裁剪，但在tidb的实现中实际  
下获取每个算子所涉及（需要的）到的列，将其传递给下层  
算子，列数是在增加的。

//project消除  
plan/rule\_eliminate\_projection.go  
projectionEliminator.optimize()

//也是根逻辑计划节点开始，先递归消除所有的孩子节点的冗余  
然后消除自身冗余的**project**逻辑计划算子  
**plan/rule\_eliminate\_projection.go**  
**projectionEliminator.eliminate()**

//构建key相关信息  
plan/plan/rule\_build\_key\_info.go  
buildKeySolver.optimize()

//构建**unique key**和**maxOneRow**属性  
先调用子孩子的**buildKeyInfo()**，从叶子节点向上传递key信息

//去相关子查询，变为join  
plan/rule\_decorrelate.go  
decorrelateSolver.optimize()

//将相关子查询的**apply**逻辑计划改写为**join**  
先重写当前节点的**apply**为**join**，再递归处理子孩子的的**apply**

//最大最小值消除  
plan/rule\_max\_min\_eliminate.go  
maxMinEliminator.optimize()

//先重写当前**min/max**，增加**sort,limit**,算子和**is not null**条件  
后递归处理子孩子节点  
**plan/rule\_max\_min\_eliminate.go**  
**maxMinEliminator.eliminateMaxMin()**

//谓词下压  
plan/rule\_predicate\_push\_down.go  
ppdSolver.optimize()

//自顶向下处理所有的逻辑计划算子，尝试将下压谓词推到下  
划算子，无法下推作为**select**算子  
下压**having**，**where**，**on**等上的谓词  
处理**join**的逻辑计划算子时会有外连接消除处理**simplifyOuterJoin**

//分区处理  
plan/rule\_partition\_processor.go  
partitionProcessor.optimize()

//剪枝谓词下压后，不需要访问的**partition**  
**plan/rule\_partition\_processor.go**  
**partitionProcessor.rewriteDataSource()**

//聚集函数下压  
plan/rule\_aggregation\_push\_down.go  
aggregationOptimizer.optimize()

//由于部分聚集函数满足可分区性，可以下压到**tikv**上执行，  
聚合各个**tikv**的子部分结果  
可下压的聚集函数包括**max,min,first\_row**  
**sum,count**,无**distinct**谓词，未实现下压的**avg**

//TOP-N下压  
plan/rule\_topn\_push\_down.go  
pushDownTopNOptimizer.optimize()

//下压**topN**和**limit**  
**p.pushDownTopN()**