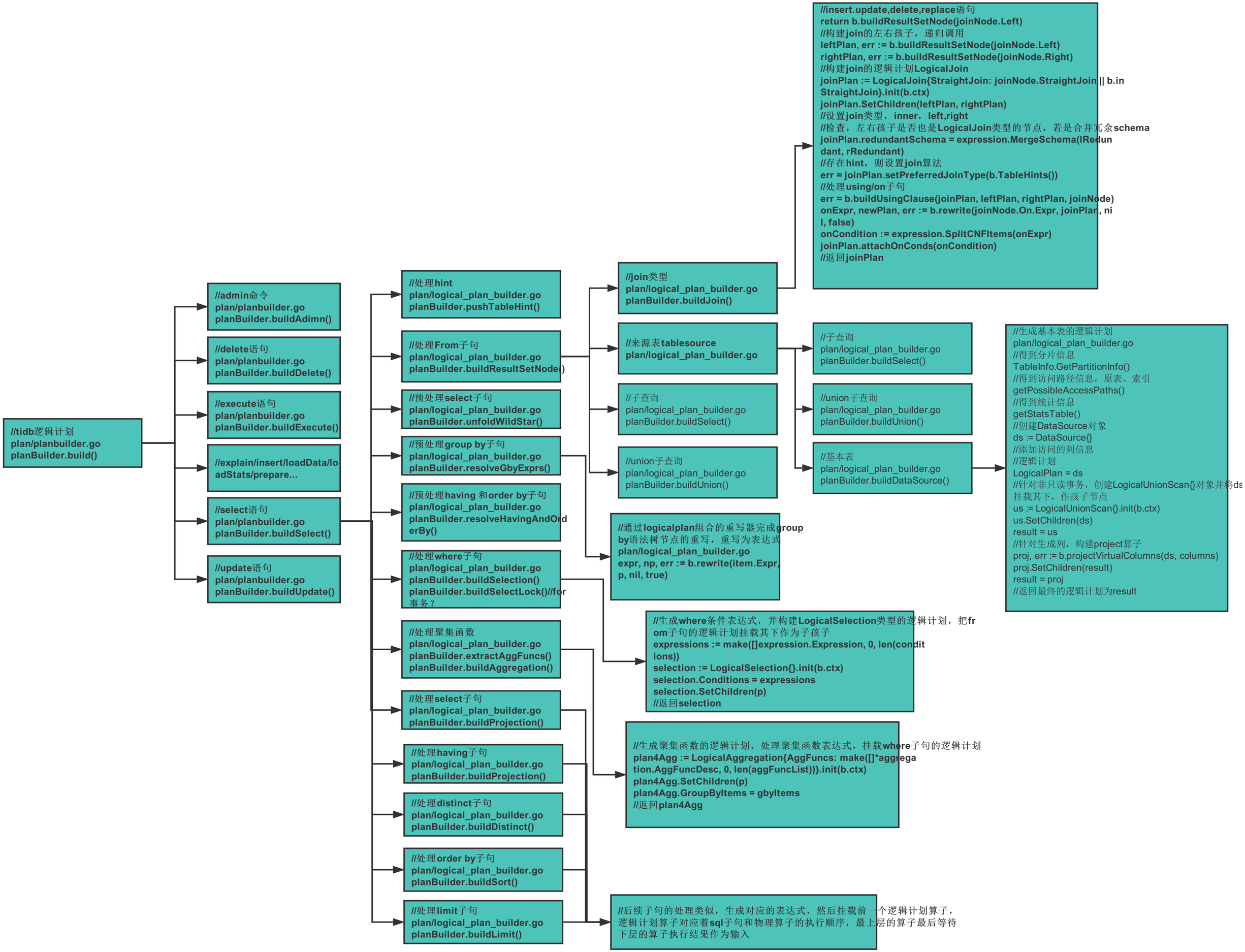
//tidb逻辑计划
plan/planbuilder.go
planBuilder.build()
//admin命令
plan/planbuilder.go
planBuilder.buildAdimn()
//delete语句
plan/planbuilder.go
planBuilder.buildDelete()
//execute语句
plan/planbuilder.go
planBuilder.buildExecute()
//explain/insert/loadData/loadStats/prepare...
//select语句
plan/planbuilder.go
planBuilder.buildSelect()
//update语句
plan/planbuilder.go
planBuilder.buildUpdate()
//处理hint
plan/logical_plan_builder.go
planBuilder.pushTableHint()
//处理From子句
plan/logical_plan_builder.go
planBuilder.buildResultSetNode()
//预处理select子句
plan/logical_plan_builder.go
planBuilder.unfoldWildStar()
//预处理group by子句
plan/logical_plan_builder.go
planBuilder.resolveGbyExprs()
//预处理having 和order by子句
plan/logical_plan_builder.go
planBuilder.resolveHavingAndOrderBy()
//处理where子句
plan/logical_plan_builder.go
planBuilder.buildSelection()
planBuilder.buildSelectLock()//for事务？
//处理聚集函数
plan/logical_plan_builder.go
planBuilder.extractAggFuncs()
planBuilder.buildAggregation()
//处理select子句
plan/logical_plan_builder.go
planBuilder.buildProjection()
//处理having子句
plan/logical_plan_builder.go
planBuilder.buildProjection()
//处理distinct子句
plan/logical_plan_builder.go
planBuilder.buildDistinct()
//处理order by子句
plan/logical_plan_builder.go
planBuilder.buildSort()
//处理limit子句
plan/logical_plan_builder.go
planBuilder.buildLimit()
//join类型
plan/logical_plan_builder.go
planBuilder.buildJoin()
//来源表tablesource
plan/logical_plan_builder.go
//子查询
plan/logical_plan_builder.go
planBuilder.buildSelect()
//union子查询
plan/logical_plan_builder.go
planBuilder.buildUnion()
//通过logicalplan组合的重写器完成group by语法树节点的重写，重写为表达式
plan/logical_plan_builder.go
expr, np, err := b.rewrite(item.Expr, p, nil, true)
//insert.update,delete,replace语句
return b.buildResultSetNode(joinNode.Left)
//构建join的左右孩子，递归调用
leftPlan, err := b.buildResultSetNode(joinNode.Left)
rightPlan, err := b.buildResultSetNode(joinNode.Right)
//构建join的逻辑计划LogicalJoin
joinPlan := LogicalJoin{StraightJoin: joinNode.StraightJoin || b.inStraightJoin}.init(b.ctx)
joinPlan.SetChildren(leftPlan, rightPlan)
//设置join类型，inner，left,right
//检查，左右孩子是否也是LogicalJoin类型的节点，若是合并冗余schema
joinPlan.redundantSchema = expression.MergeSchema(lRedundant, rRedundant)
//存在hint，则设置join算法
err = joinPlan.setPreferredJoinType(b.TableHints())
//处理using/on子句
err = b.buildUsingClause(joinPlan, leftPlan, rightPlan, joinNode)
onExpr, newPlan, err := b.rewrite(joinNode.On.Expr, joinPlan, nil, false)
onCondition := expression.SplitCNFItems(onExpr)
joinPlan.attachOnConds(onCondition)
//返回joinPlan
//子查询
plan/logical_plan_builder.go
planBuilder.buildSelect()
//union子查询
plan/logical_plan_builder.go
planBuilder.buildUnion()
//基本表
plan/logical_plan_builder.go
planBuilder.buildDataSource()
//生成基本表的逻辑计划
plan/logical_plan_builder.go
//得到分片信息
TableInfo.GetPartitionInfo()
//得到访问路径信息，原表、索引
getPossibleAccessPaths()
//得到统计信息
getStatsTable()
//创建DataSource对象
ds := DataSource{}
//添加访问的列信息
//逻辑计划
LogicalPlan = ds
//针对非只读事务，创建LogicalUnionScan{}对象并将ds挂载其下，作孩子节点
us := LogicalUnionScan{}.init(b.ctx)
us.SetChildren(ds)
result = us
//针对生成列，构建project算子
proj, err := b.projectVirtualColumns(ds, columns)
proj.SetChildren(result)
result = proj
//返回最终的逻辑计划为result
//生成where条件表达式，并构建LogicalSelection类型的逻辑计划，把from子句的逻辑计划挂载其下作为子孩子
expressions := make([]expression.Expression, 0, len(conditions))
selection := LogicalSelection{}.init(b.ctx)
selection.Conditions = expressions
selection.SetChildren(p)
//返回selection
//生成聚集函数的逻辑计划，处理聚集函数表达式，挂载where子句的逻辑计划
plan4Agg := LogicalAggregation{AggFuncs: make([]*aggregation.AggFuncDesc, 0, len(aggFuncList))}.init(b.ctx)
plan4Agg.SetChildren(p)
plan4Agg.GroupByItems = gbyItems
//返回plan4Agg
//后续子句的处理类似，生成对应的表达式，然后挂载前一个逻辑计划算子，逻辑计划算子对应着sql子句和物理算子的执行顺序，最上层的算子最后等待下层的算子执行结果作为输入