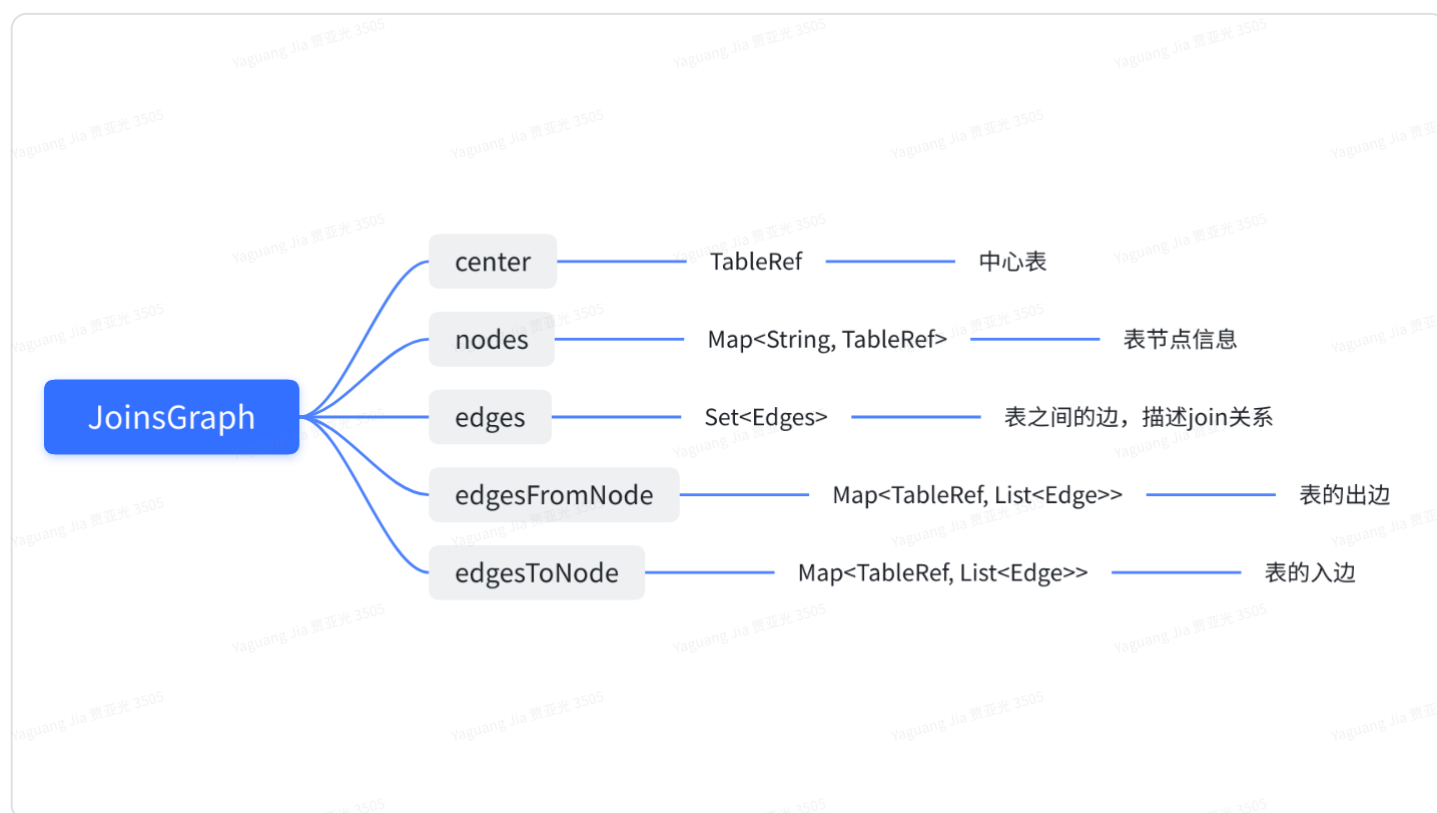


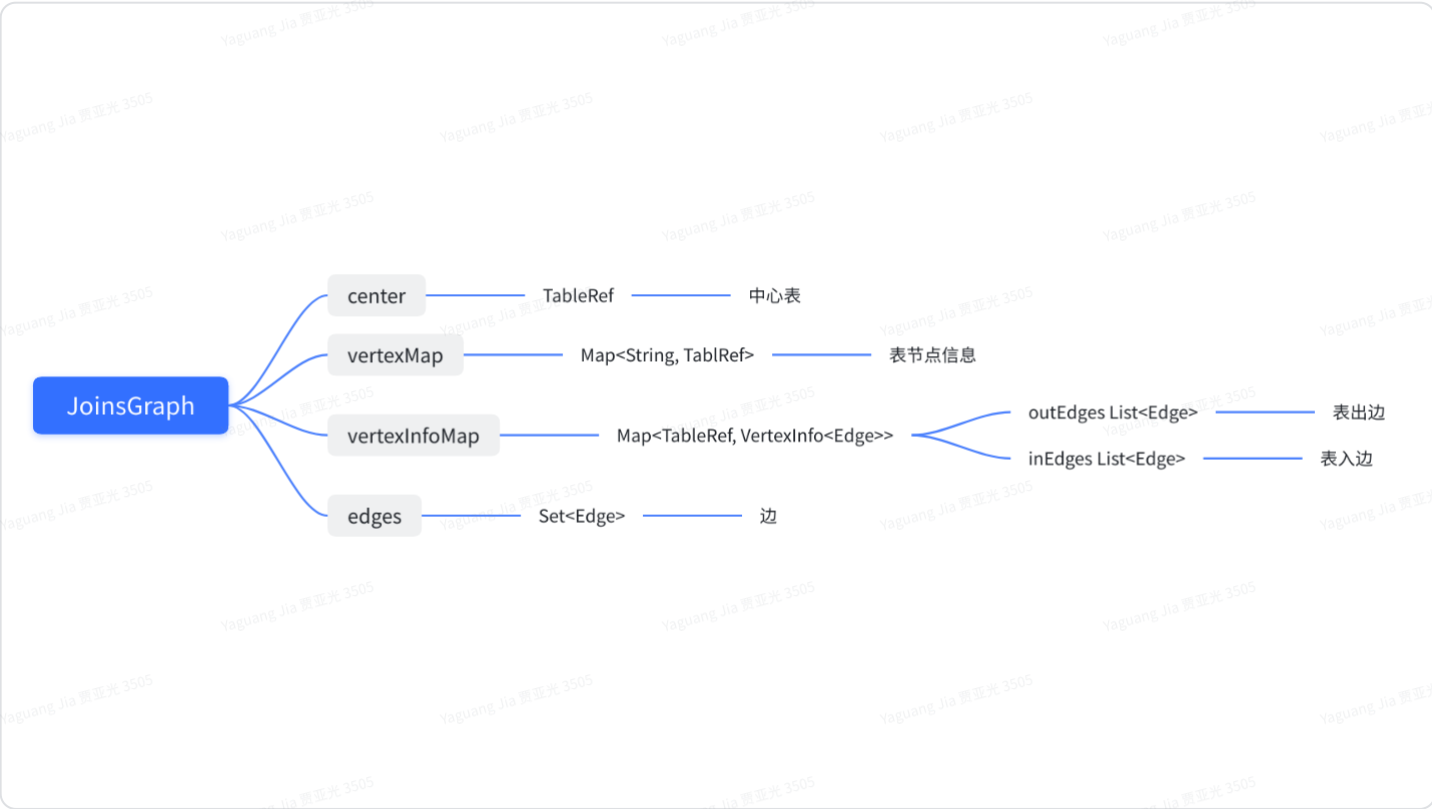
# KYLIN-5521 JoinsGraph optimization: Query SQL association order change causes the model to fail to hit

## JoinsGraph 数据结构变更

1. center 中心表和 nodes 表节点信息结构保持不变
2. edges 对于 inner 或者 innerOrLeft join，会生成两条PK和FK相反的两条边，并标记edge是交换过的(swapJoin=true)
3. 表的出边和入边统一用vertexInfoMap表示，vertexInfo中有表示出入边的两个List

## 旧版

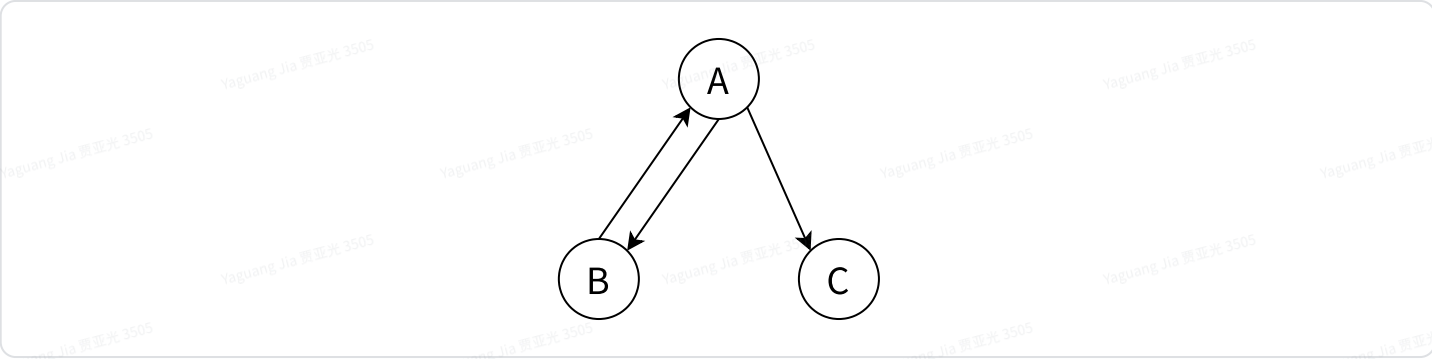




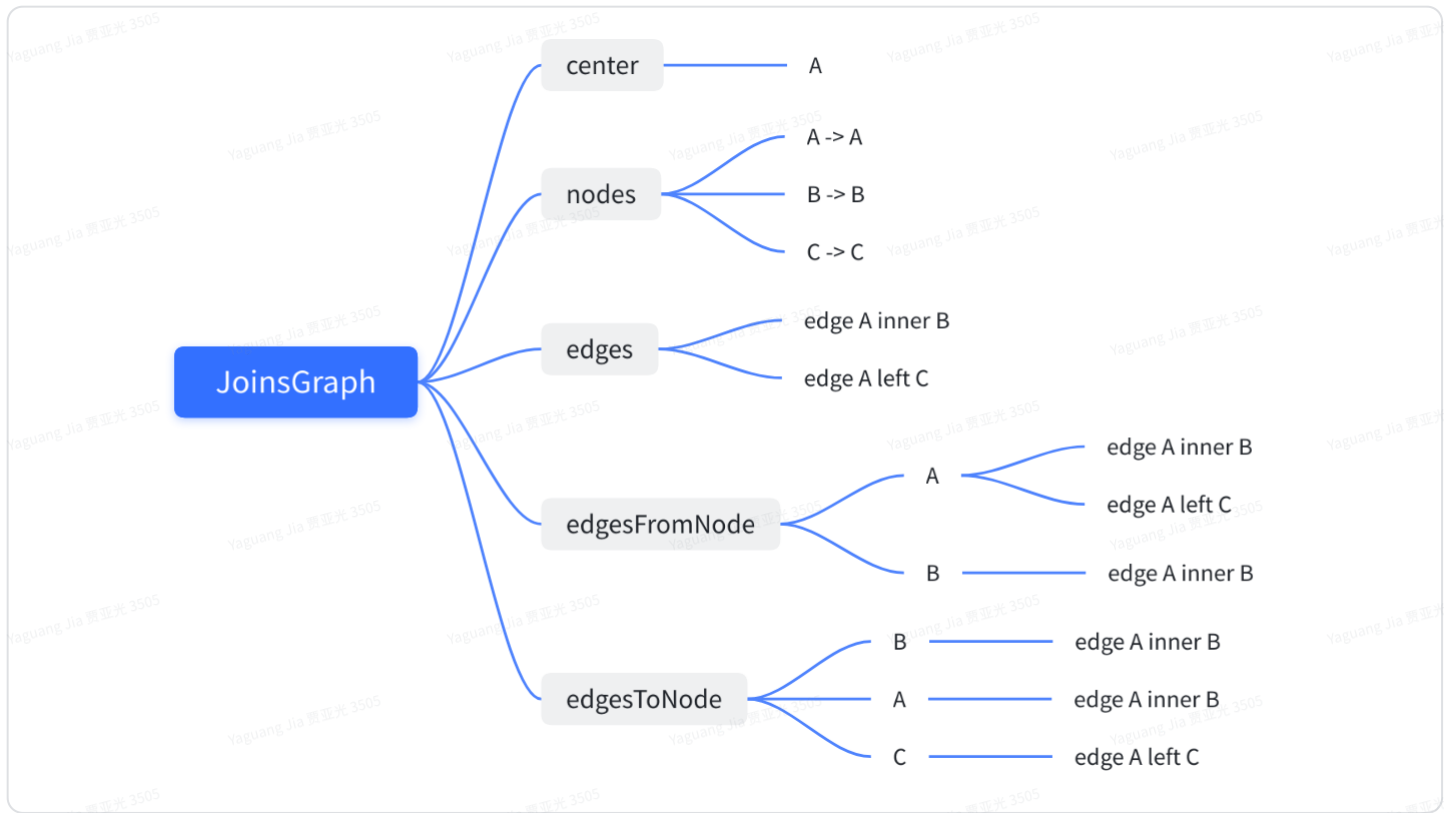
举例

Join关系如下

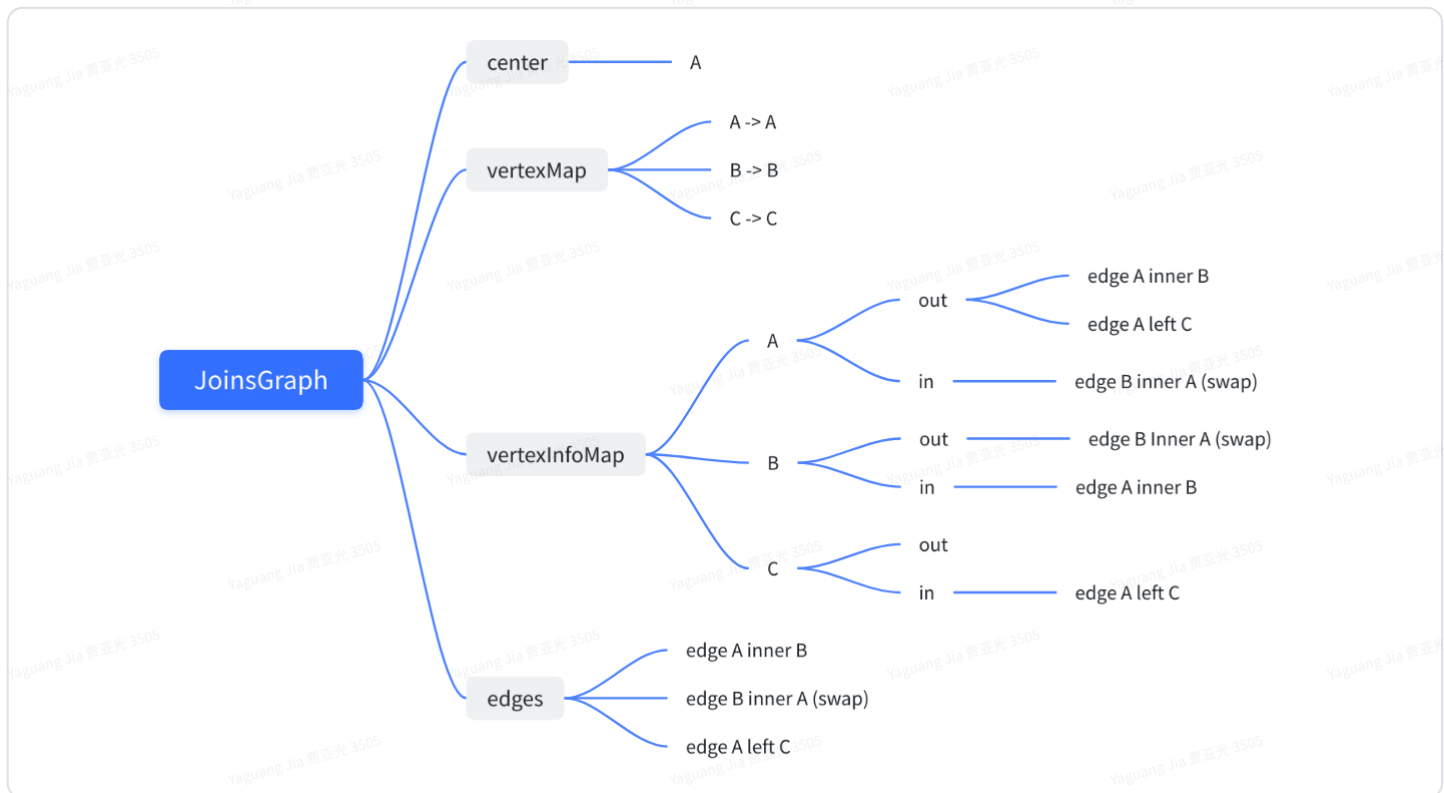
```
1 A inner join B
2 A left join C
```



旧版JoinsGraph



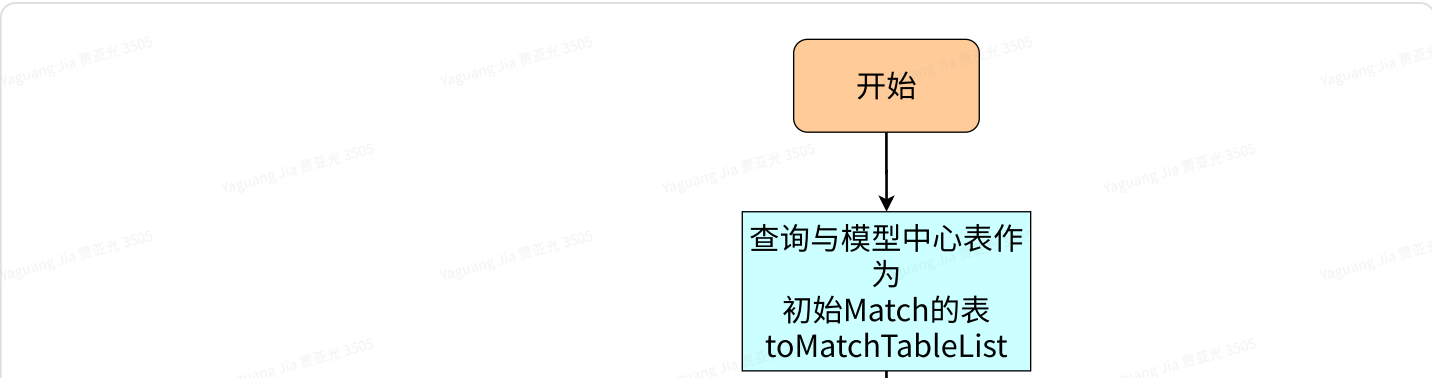
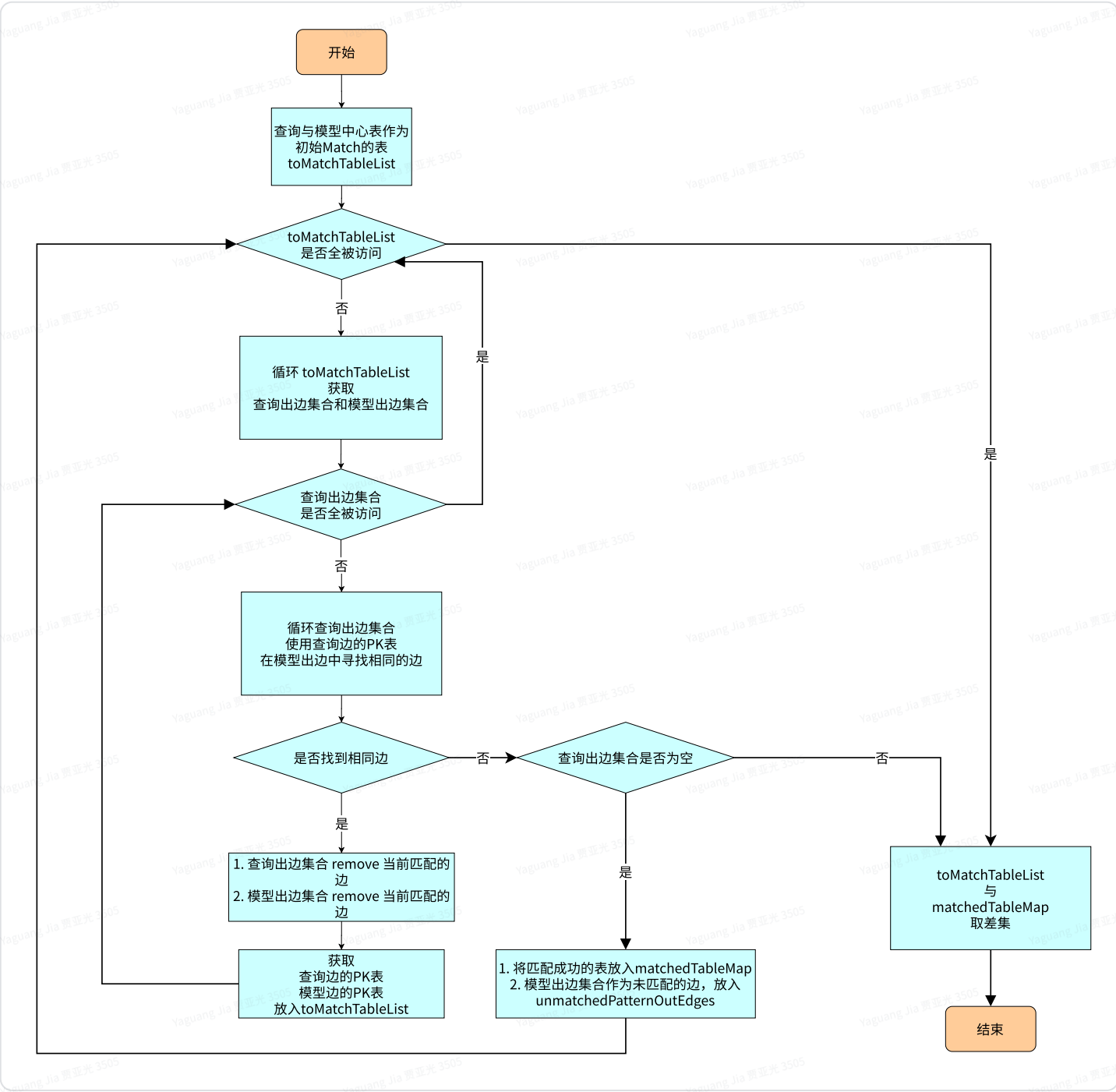
## 新版JoinsGraph

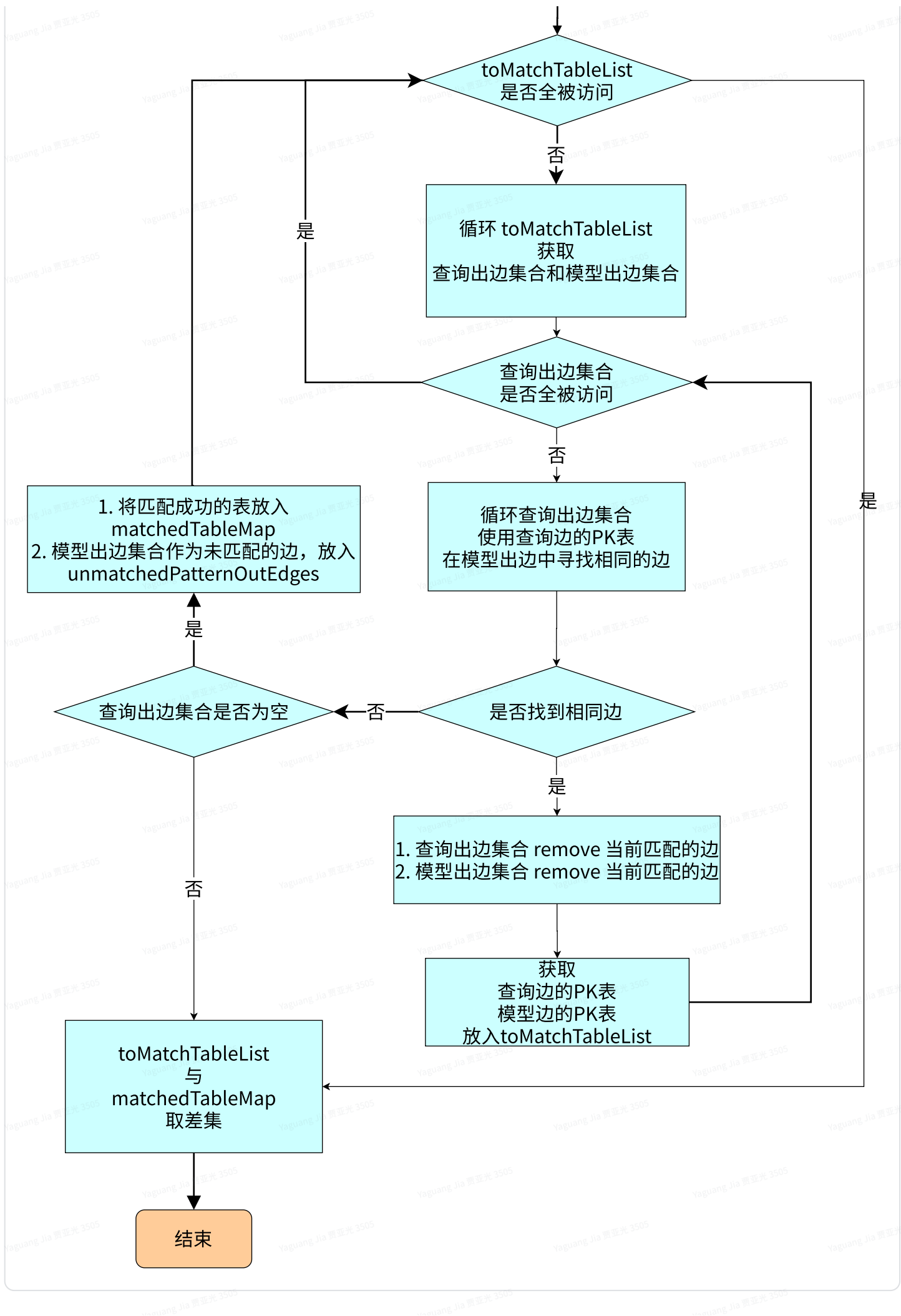


## innerMatch 逻辑改造

找出查询和模型的中间表，存入 **toMatchTableList**

将查询 JoinsGraph 与一个模型 JoinsGraph 匹配。所有匹配的表将被放入参数 matchedTableMap 中。不匹配的出边将被放入参数 unmatchedPatternOutEdges 中。





## 详细逻辑

1. 找到Query与Model的相同中心表，由此开始遍历两个Graph
  - a. 遍历集合为 `toMatchTableList`，一边遍历一边新增待匹配的表节点
2. 以Query节点为主，获取查询OutEdge集合，在模型Graph中寻找能匹配的OutEdge
  - a. 查询Edge是 `leftOrInnerJoin`，且模型的出边为空时，这条边直接匹配成功，进入下一个循环
  - b. 边的PK表相同
  - c. 边相同
  - d. 上述条件匹配到的边数量为1时，直接返回
  - e. 匹配到的边数量不为1时，看边的PK表的出边数量是否相同
  - f. 其他返回NULL
3. 匹配到的边
  - a. 为NULL时，匹配失败返回
  - b. 不为NULL时，表示查询Edge与模型Edge已匹配
    - i. 查询OutEdge集合与模型OutEdge集合，均remove当前匹配成功的Edge（成功则remove，不成功则保留）
    - ii. 将查询Edge与模型Edge的PK表加入 `toMatchList`，将在下次循环时匹配
4. 上述查询OutEdge遍历完毕后
  - a. 判断查询OutEdge是否全匹配完毕(集合是否为空)
  - b. 如全匹配完毕
    - i. `matchedTableMap` 添加当前待匹配的节点
    - ii. `unmatchedOutEdgesOfPattern` 添加全部 模型OutEdge集合（匹配成功的会被remove，所以剩下的就是unmatched）
5. `toMatchTableList` 与 `matchedTableMap` 取差集
  - a. 取差集后结果为空，并且Query的所有节点均被访问(`allMatchedTableMap` 大小与顶点数量是否相等)，则表示成功匹配。
6. 校验 `unmatchedOutEdgesOfPattern`
  - a. `unmatchedOutEdgesOfPattern` 为空，表示精确匹配
  - b. `unmatchedOutEdgesOfPattern` 不为空，但是所有的边都是LeftJoin，表示精确匹配
  - c. `unmatchedOutEdgesOfPattern` 不为空，但是 `matchPartial = true`，表示部分匹配

d. 满足上述三个条件之一，即可认为匹配成功。