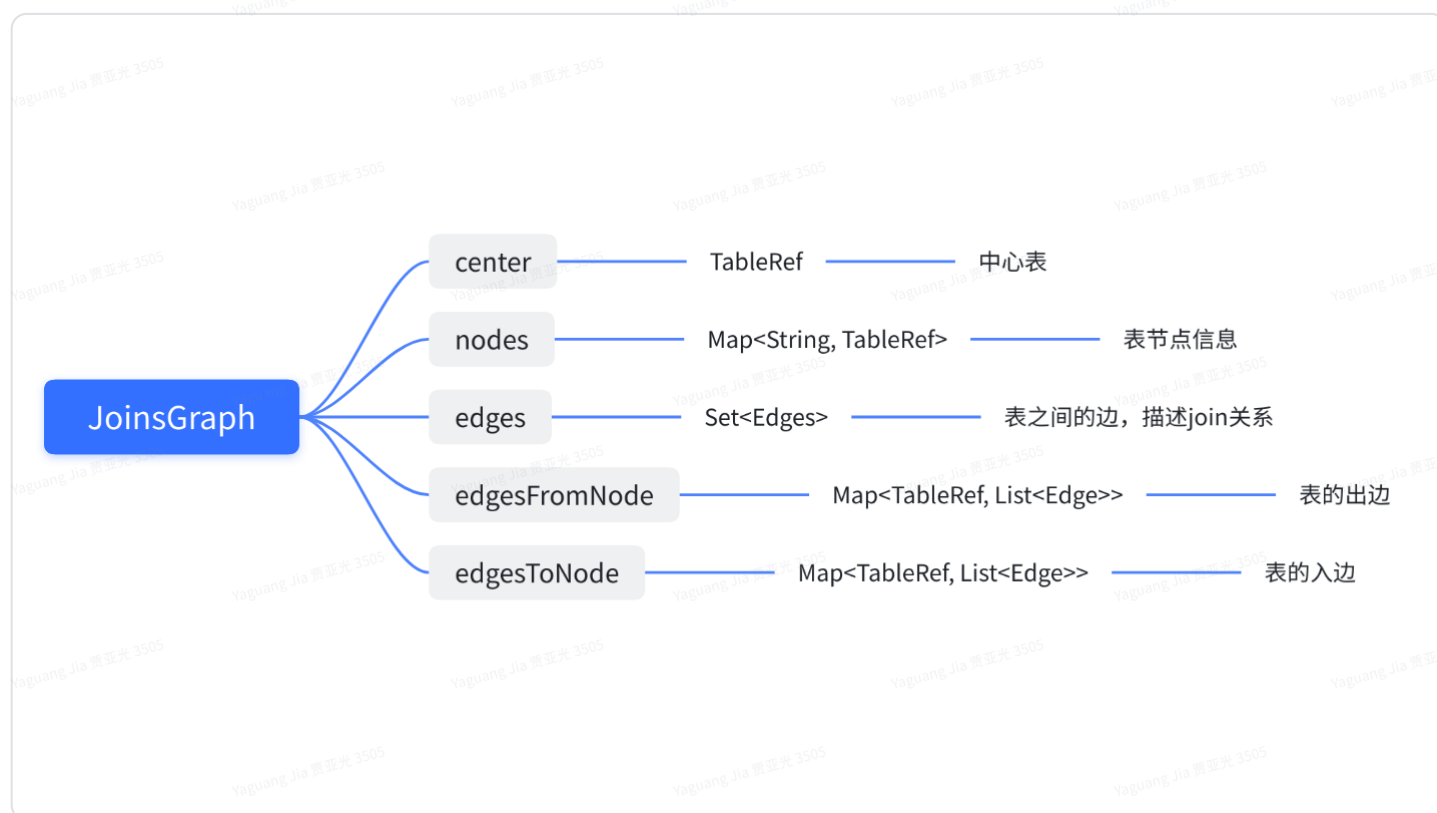


# KYLIN-5521 JoinsGraph optimization: Query SQL association order change causes the model to fail to hit

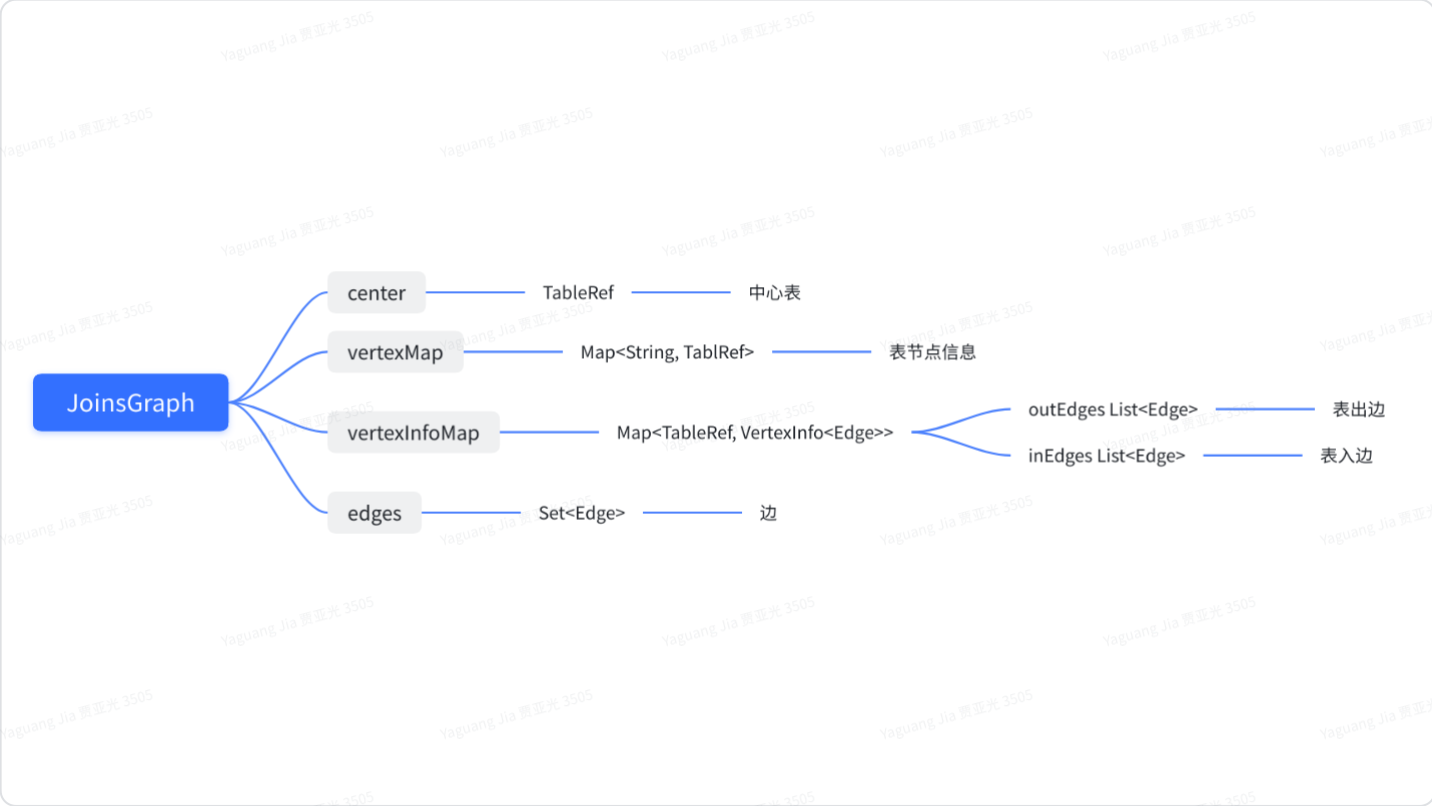
## JoinsGraph data structure changes

1. Center center table and nodes table node information structure remains unchanged
2. Edges For inner or innerOrLeft join, generate two opposite edges of PK and FK, and mark the edges as swapped (swapJoin = true)
3. The outbound and inbound edges of the table are uniformly represented by vertexInfoMap. There are two Lists in vertexInfo that represent the incoming and outgoing edges.

## Old version



# New version



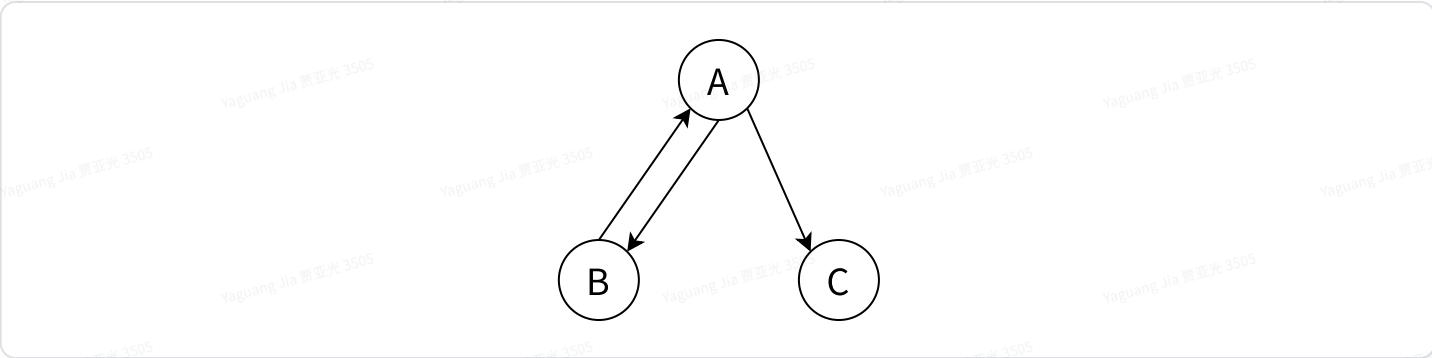
# Example

Join the relationship as follows

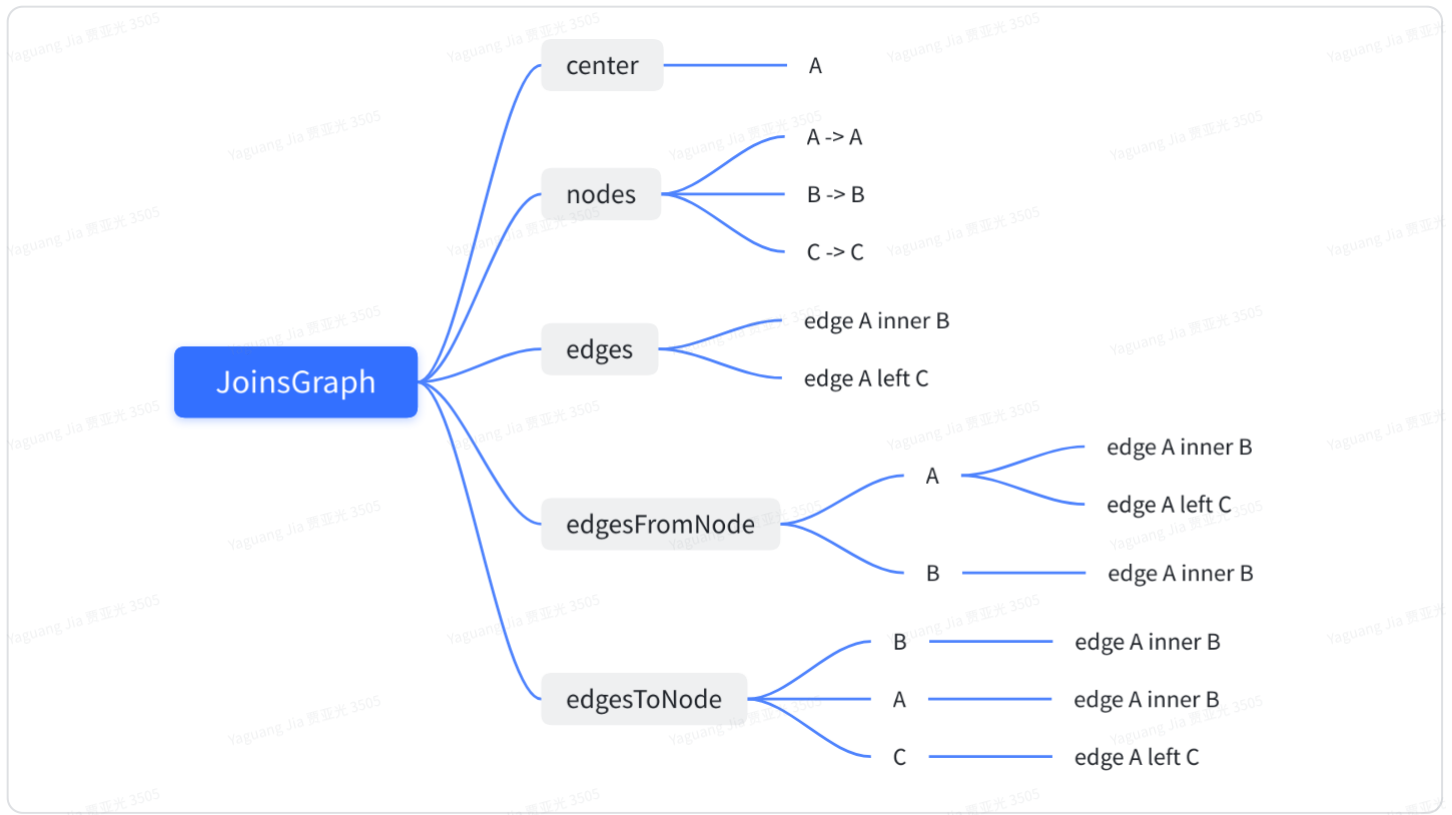
- 1

A inner join B
- 2

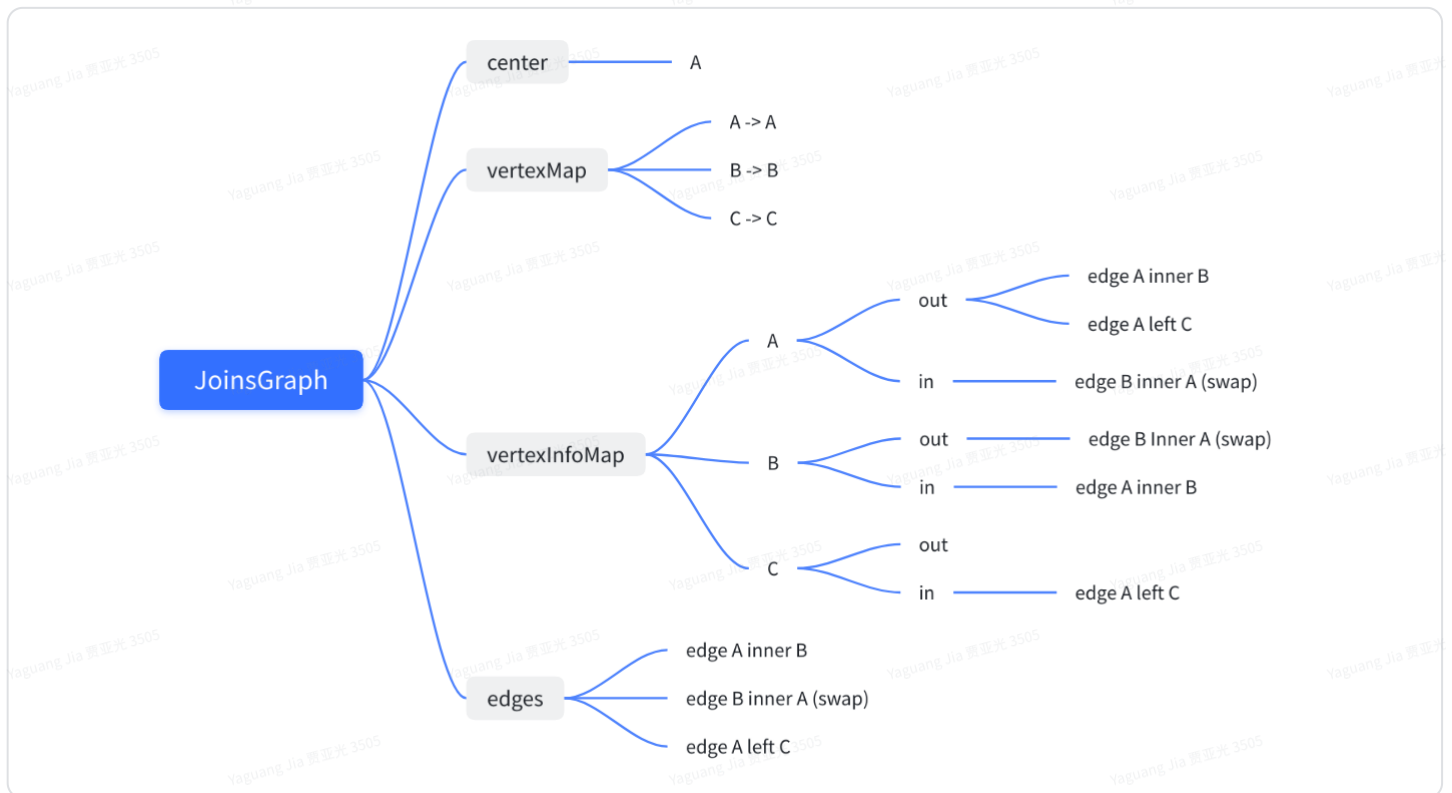
A left join C



# 旧版JoinsGraph



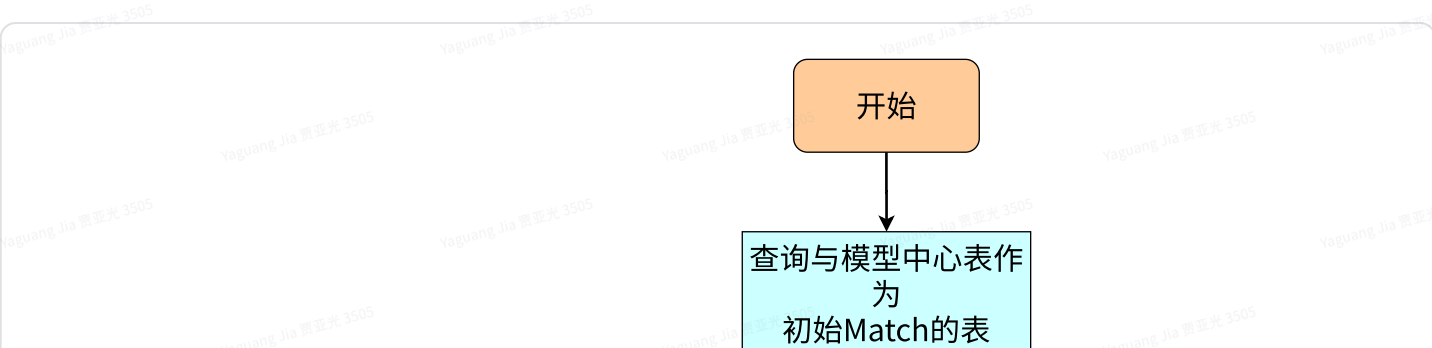
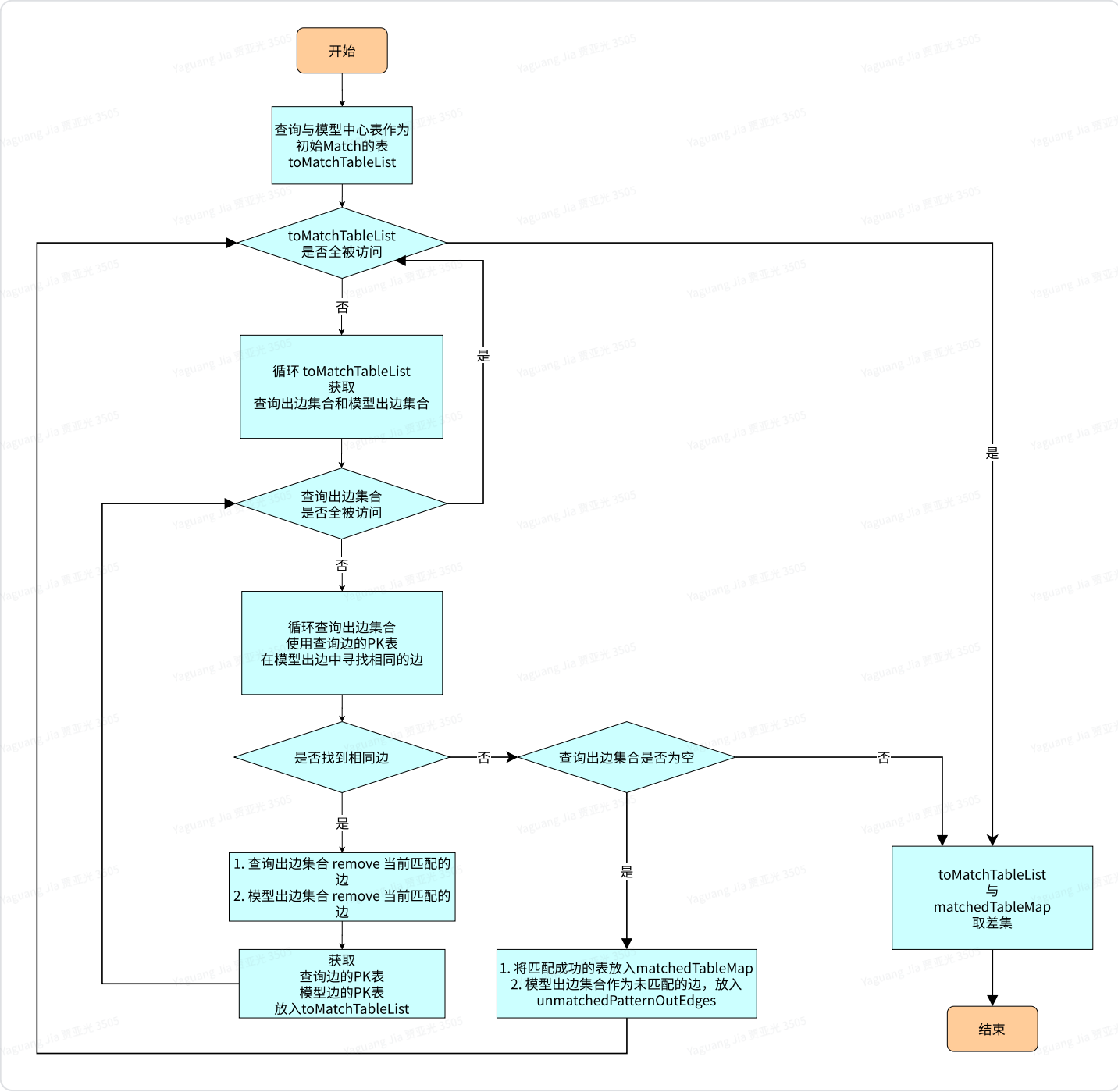
## 新版JoinsGraph

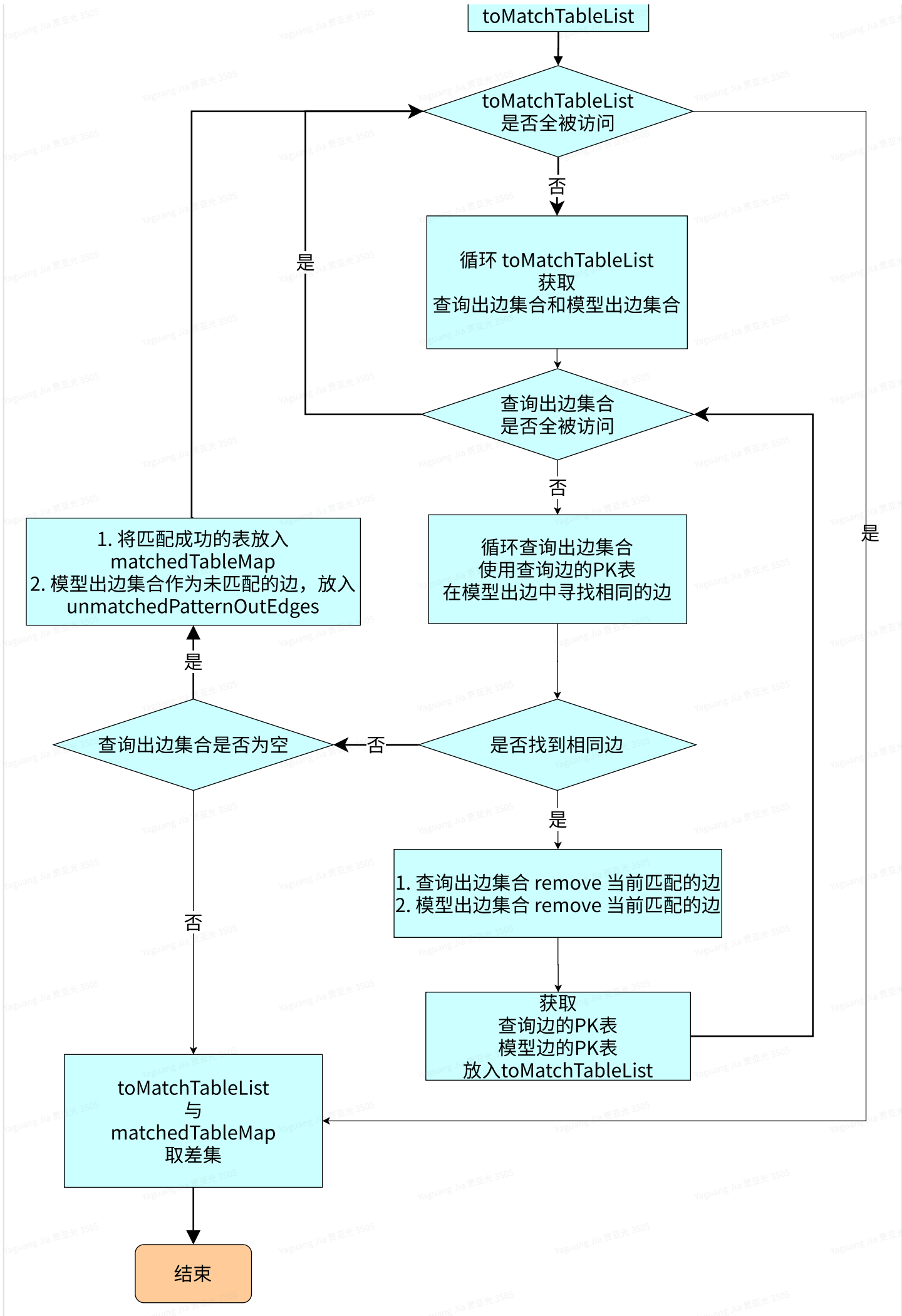


## innerMatch logic transformation

Find the intermediate table of query and model and save it into **toMatchTableList**

Matches the query `JoinsGraph` to a model `JoinsGraph` . All matching tables will be placed in the parameter `matchedTableMap` . Mismatched outlines will be placed in the parameter `unmatchedPatternOutEdges` .





## Detailed logic

- Find the same central table of Query and Model, and start traversing the two graphs
  - Traverse the collection to `toMatchTableList` , and **add table nodes to be matched while traversing**
- Based on the Query node, get the query OutEdge collection, and find the matching OutEdge in the model Graph.
  - When the query Edge is `leftOrInnerJoin` , and the outbound of the model is **empty, this edge directly matches successfully and enters the next loop**
  - The PK table on the side is the same.
  - Edge is the same
  - When the number of edges matched by the above conditions is 1, directly return
  - When the number of matched edges is not 1, see if the number of outgoing edges in the PK table of the edges is the same.
  - Others return NULL
- Edge to match
  - When NULL, the match fails and returns
  - If not NULL, it means that the query Edge matches the model Edge
    - Query OutEdge collection and model OutEdge collection, remove the current matching Edge successfully ( **remove if successful, keep if unsuccessful** )
    - Add the query Edge and the PK table of the model Edge to the `toMatchList` , **which will match on the next loop**
- After the above query OutEdge traversal is completed
  - Determine whether the query OutEdge is fully matched ( **whether the collection is empty** )
  - If all matches are completed
    - `matchedTableMap Add` the current node to be matched
    - `unmatchedOutEdgesOfPattern Add` all, model OutEdge collection (successful matches will be removed, so the rest is unmatched)
- `toMatchTableList` and `matchedTableMap` take the difference set

- a. After taking the difference set, the result is empty, and all nodes of Query are visited ( **whether the allMatchedTableMap size is equal to the number of vertices** ), it means a successful match.

6. Verify `unmatchedOutEdgesOfPattern`

- a. `unmatchedOutEdgesOfPattern` is empty, indicating an exact match
- b. `unmatchedOutEdgesOfPattern` is not empty, but all edges are LeftJoin, indicating an exact match
- c. `unmatchedOutEdgesOfPattern` is not empty, but `matchPartial = true` , indicating a partial match
- d. If one of the above three conditions is met, the match is considered successful.