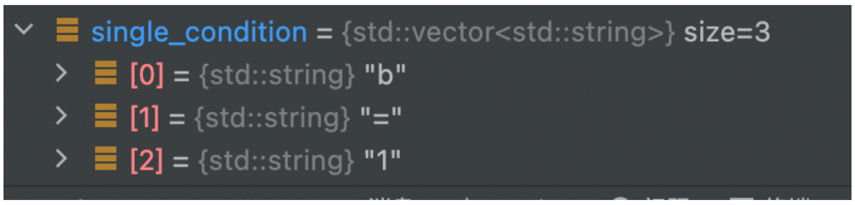
Condition

单个Condition获取，传入kNodeCompareOperator

```
vector<string> GetSingleCondition(SyntaxNode* node){
  ASSERT(node->type_ == kNodeCompareOperator,"ERROR: Can't get condition");
  // op:'<','>','=','not','is'
  string op = node -> val_;
  string col_name = node -> child_ -> val_;
  string value = "";
  auto value_node = node -> child_ -> next_;
  //value or null
  if(value_node -> type_ == kNodeNull){
    value = "null";
  }else{
    value = value_node -> val_;
  }
  vector<string> condition = {col_name,op,value};
  return condition;
}
```

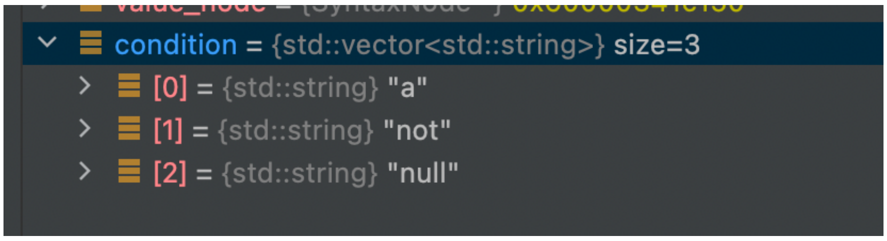
where b = 1;

返回的结果是：



```
single_condition = {std::vector<std::string>} size=3
  [0] = {std::string} "b"
  [1] = {std::string} "="
  [2] = {std::string} "1"
```

where a not null;



```
condition = {std::vector<std::string>} size=3
  [0] = {std::string} "a"
  [1] = {std::string} "not"
  [2] = {std::string} "null"
```

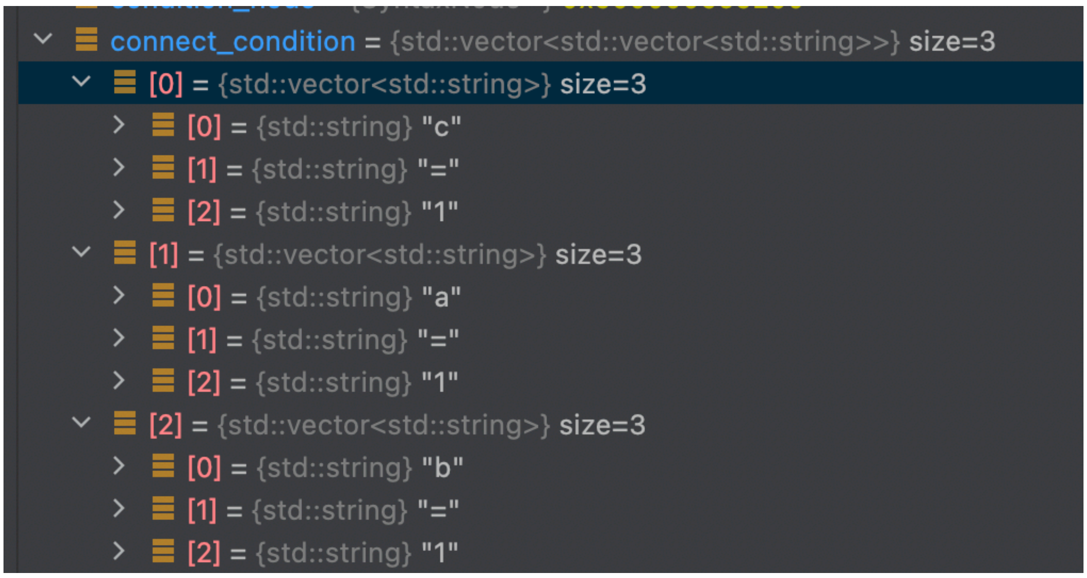
connected-condition(已合并至第二个函数)

有多个and，连接多个条件

返回vector<vector< string >>

connect的next

a = 1 and b = 1 and c =1



```
connect_condition = {std::vector<std::vector<std::string>>} size=3
  [0] = {std::vector<std::string>} size=3
    [0] = {std::string} "c"
    [1] = {std::string} "="
    [2] = {std::string} "1"
  [1] = {std::vector<std::string>} size=3
    [0] = {std::string} "a"
    [1] = {std::string} "="
    [2] = {std::string} "1"
  [2] = {std::vector<std::string>} size=3
    [0] = {std::string} "b"
    [1] = {std::string} "="
    [2] = {std::string} "1"
```

GetAllConditions（你所调用的函数）

需要：传入的node种类为kNodeConditions , conditions 为获取到条件的容器。

void ExecuteEngine::GetAllConditions(SyntaxNode* node,vector<vector<vector>> & conditions)

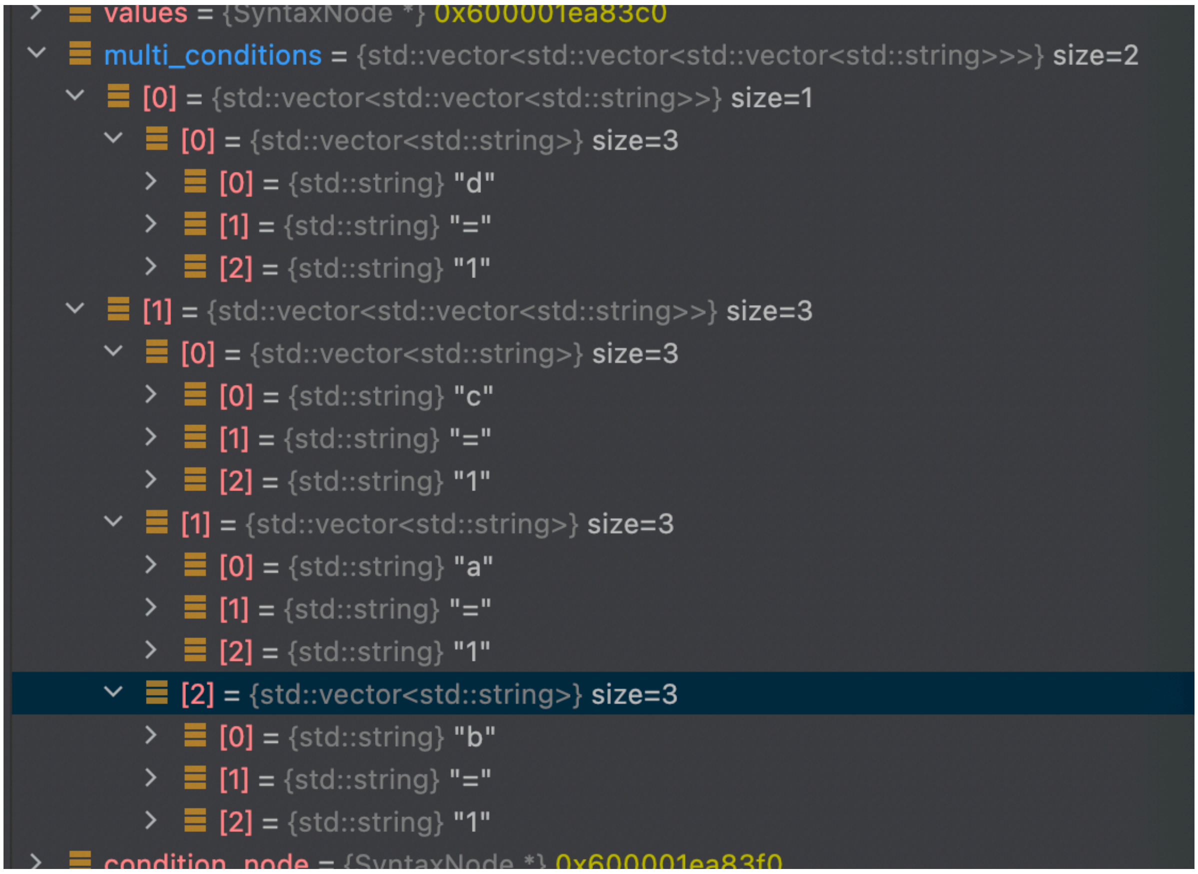
or作为connect-condition的分割，采取的方法是遍历语法树的child的节点，将新的connect放至vector尾部，同时其next为operator节点，可以生成的新的condition，压入condition的那个vecotr尾部。

采取遍历的策略：

- 遇到and，连接condition
- 遇到or则continue，生成connect condition
- 将connectcondition压入conditions

```
int j=0,i=0;
// or spilt; and union
//like ( x and y) |or| ( z and w and y) || (u)
for(;i < (int)condition_node.size();){
  vector<vector<string>> connect_condition;
  vector<string>first_condition = GetSingleCondition(condition_node[i++]);
  connect_condition.push_back(first_condition);
  while(j<(int)connectors.size() && connectors[j]=="and" ){
    vector<string>next_condition = GetSingleCondition(condition_node[i++]);
    connect_condition.push_back(next_condition);
    j++;
  }
  conditions.push_back(connect_condition);
  if(j<(int)connectors.size() && connectors[j]=="or" )j++;
}
```

update t1 set a=1 where a=1 and b=1 and c=1 or d=1;



```
>  values = {SyntaxNode *} 0x600001ea83c0
v  multi_conditions = {std::vector<std::vector<std::vector<std::string>>>} size=2
   v  [0] = {std::vector<std::vector<std::string>>} size=1
      v  [0] = {std::vector<std::string>} size=3
         >  [0] = {std::string} "d"
         >  [1] = {std::string} "="
         >  [2] = {std::string} "1"
   v  [1] = {std::vector<std::vector<std::string>>} size=3
      v  [0] = {std::vector<std::string>} size=3
         >  [0] = {std::string} "c"
         >  [1] = {std::string} "="
         >  [2] = {std::string} "1"
      v  [1] = {std::vector<std::string>} size=3
         >  [0] = {std::string} "a"
         >  [1] = {std::string} "="
         >  [2] = {std::string} "1"
      v  [2] = {std::vector<std::string>} size=3
         >  [0] = {std::string} "b"
         >  [1] = {std::string} "="
         >  [2] = {std::string} "1"
>  condition_node = {SyntaxNode *} 0x600001ea83f0
```