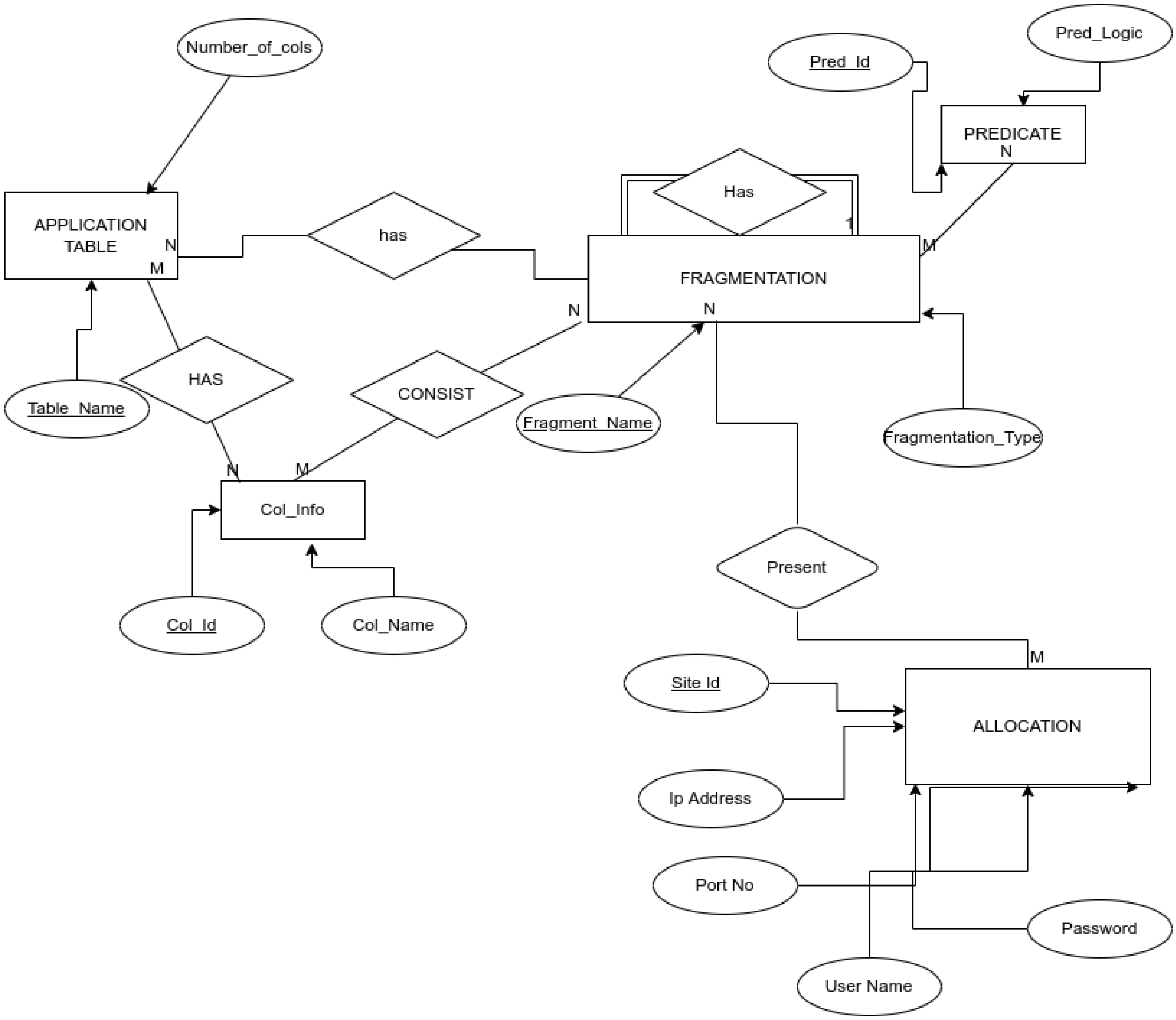
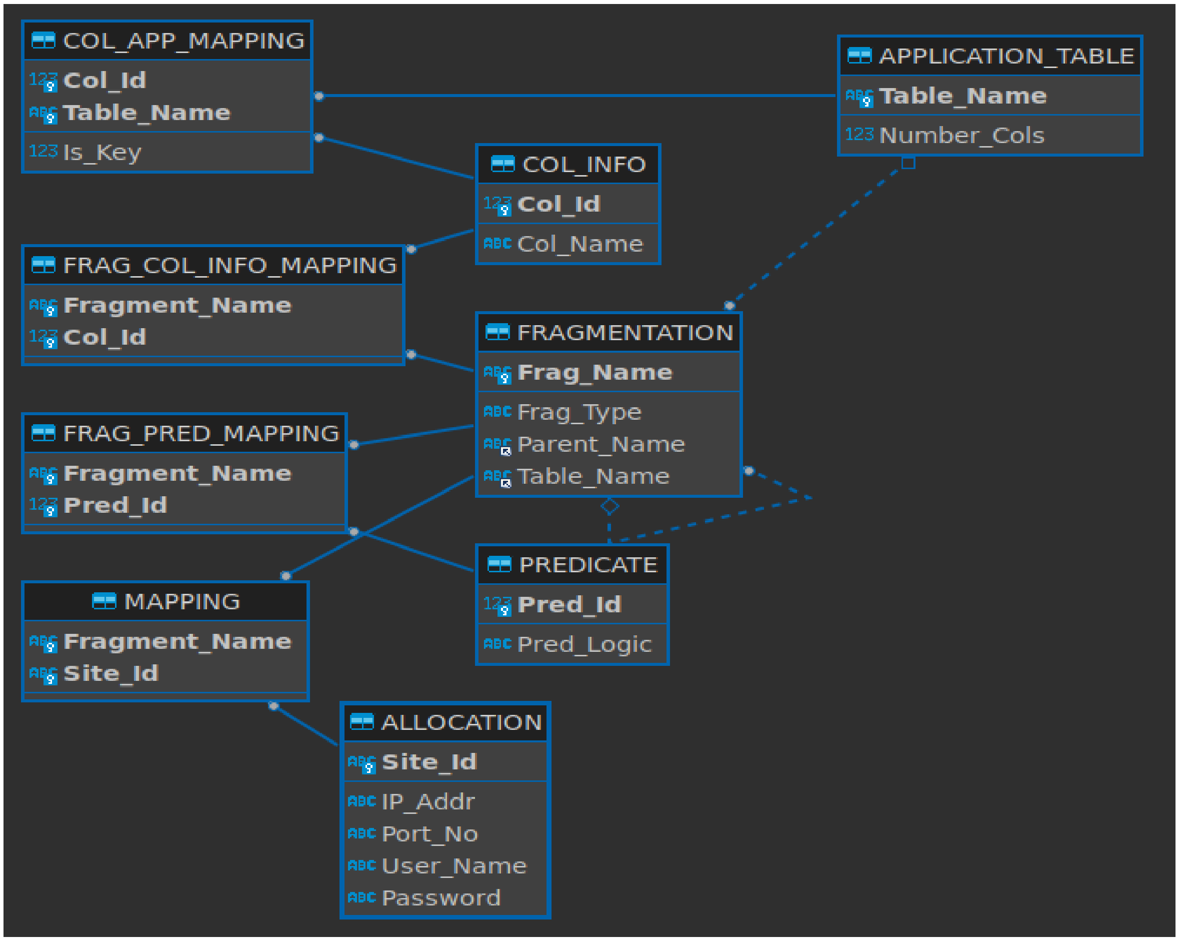
**Project Phase 2**
**Team : Samurai**

**Sai Vishwak Gangam 2020202006**
**Varun Nambigari 2020201079**

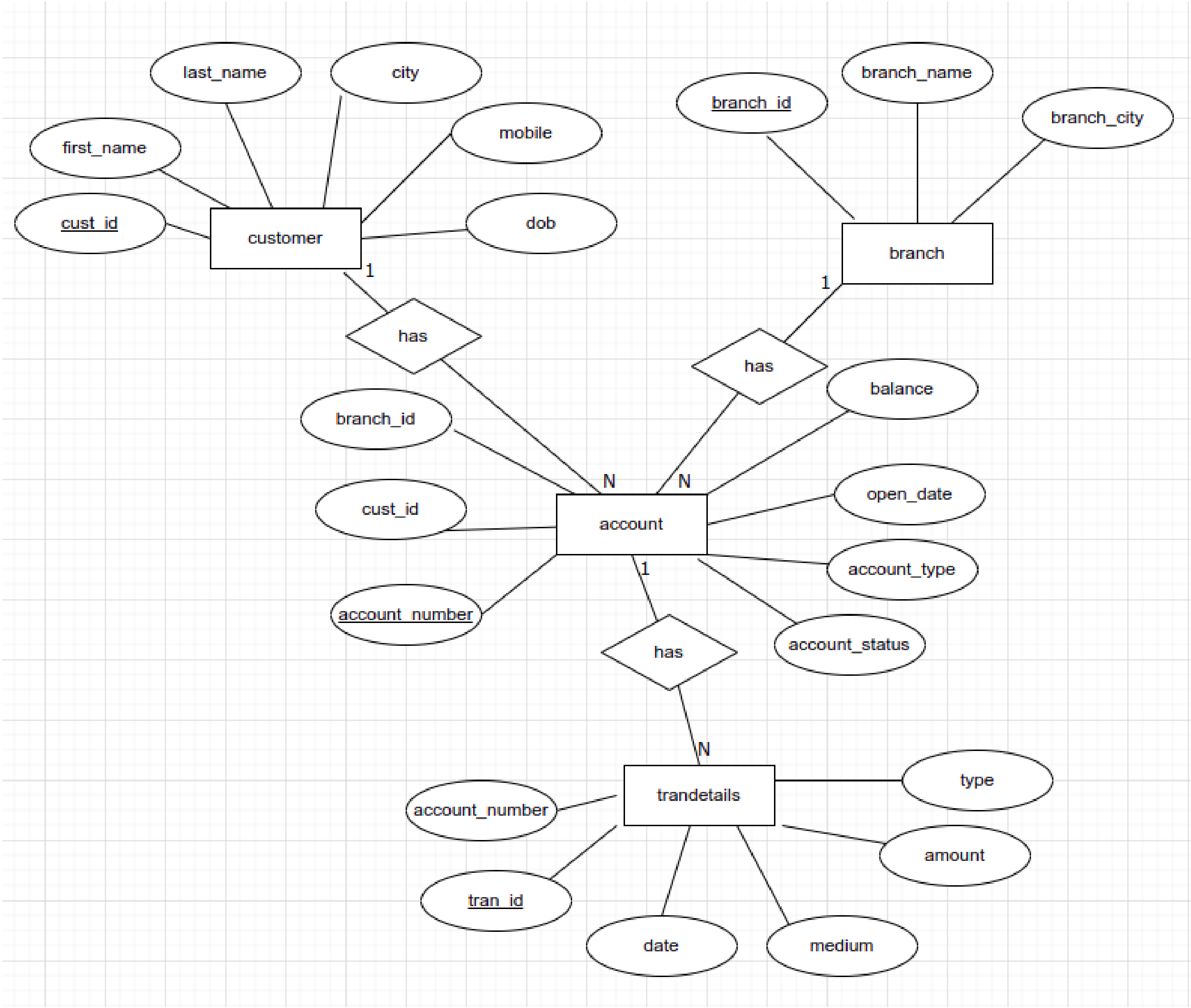
SYSTEM CATALOG (ER DIAGRAM)



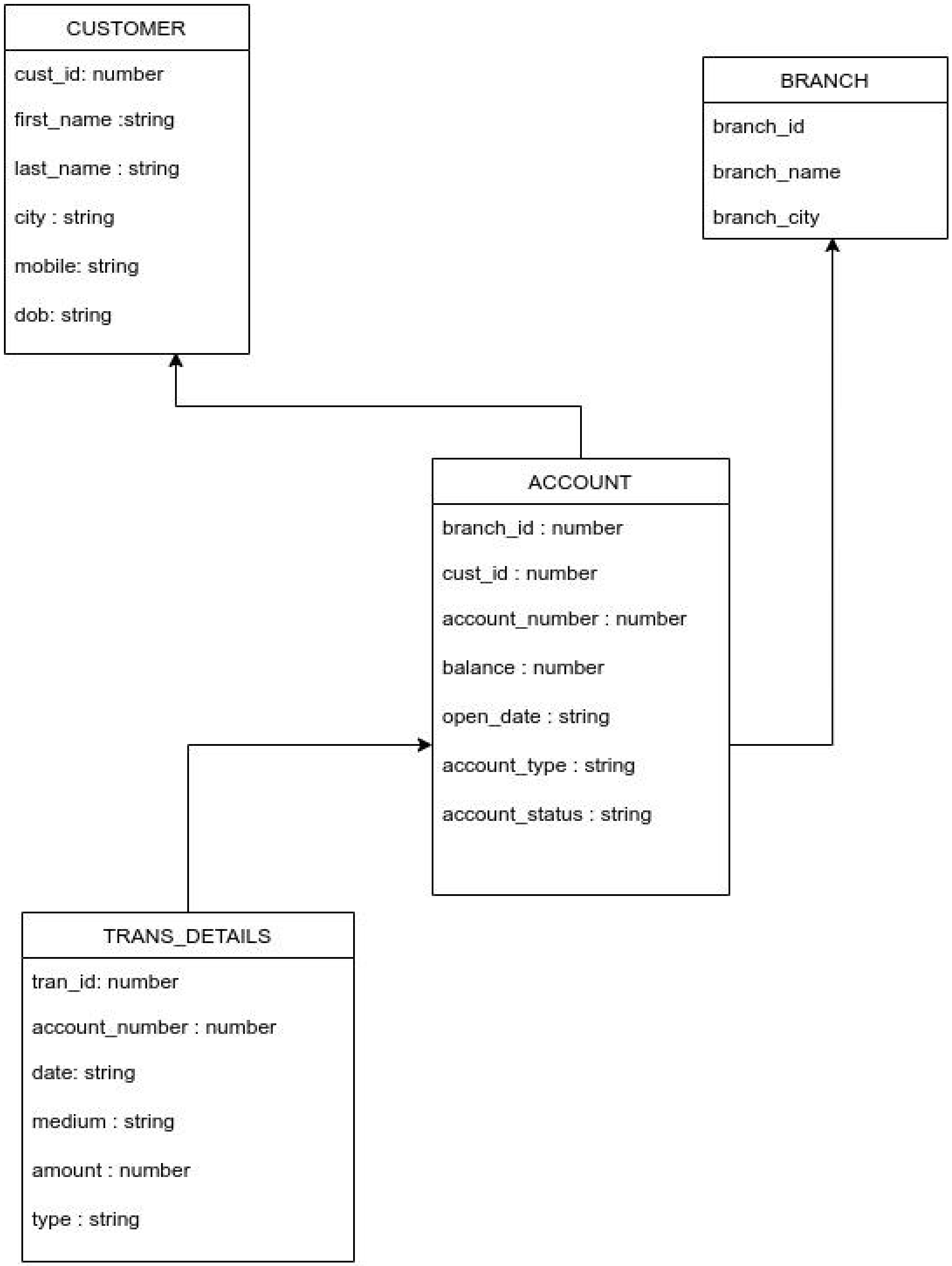
SYSTEM CATALOG( RELATIONAL MODEL)



## ER DIAGRAM of Application :



## Relational Model :


CUSTOMER
cust_id: number
first_name :string
last_name : string
city : string
mobile: string
dob: string
BRANCH
branch_id
branch_name
branch_city
ACCOUNT
branch_id : number
cust_id : number
account_number : number
balance : number
open_date : string
account_type : string
account_status : string
TRANS_DETAILS
tran_id: number
account_number : number
date: string
medium : string
amount : number
type : string

## DATA STRUCTURES USED FOR QUERY DECOMPOSITION

We used tree data structure maintaining two types of nodes : Intermediate Node and Leaf Node.

The structure of Intermediate Node is as follows:

```
class Node:
   def __init__(self, node_type, operation, operands):
       # left child
       self.left = None
       # right child
       self.right = None
       # parent node
       self.parent = None
       # data initialization
       self.node_type = node_type
       self.operation = operation
       self.operands = operands
```

The structure of Leaf Node is as follows:

```
class LeafNode:
   def __init__(self, relation_name):
       # left child
       self.left = None
       # right child
       self.right = None
       # parent node
       self.parent = None
       # node type
       self.node_type="leaf"
       # data initialization
       self.relation_name = relation_name
```

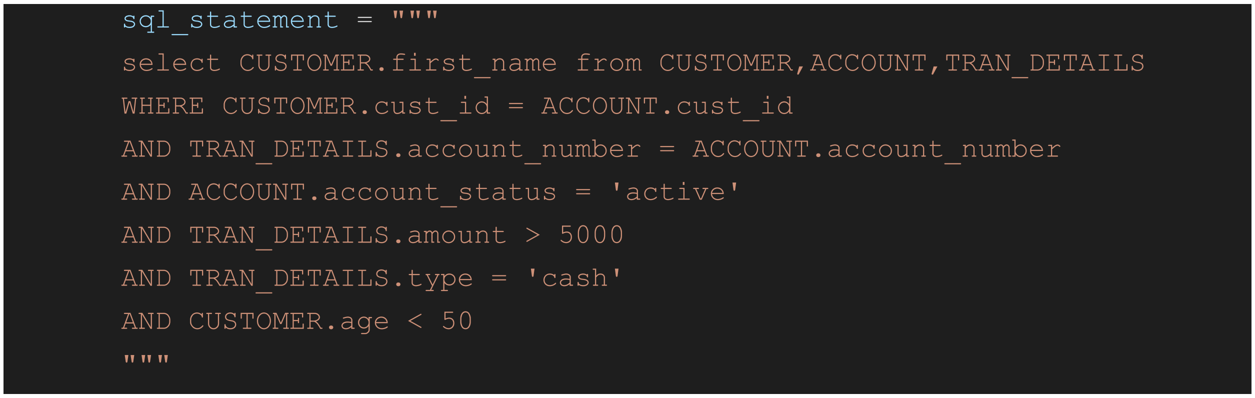
## ALGORITHM FOLLOWED:

**Step 1:** Used Mozilla Sql Parser for parsing the sql query and o/p is in JSON Format.

**Step 2:** Build an initial tree by taking json format tokenized query as input.
**Step 3:** Performed optimization of the query tree by pushing necessary select attributes down (near the relation) and pushing up all the necessary project attributes up.

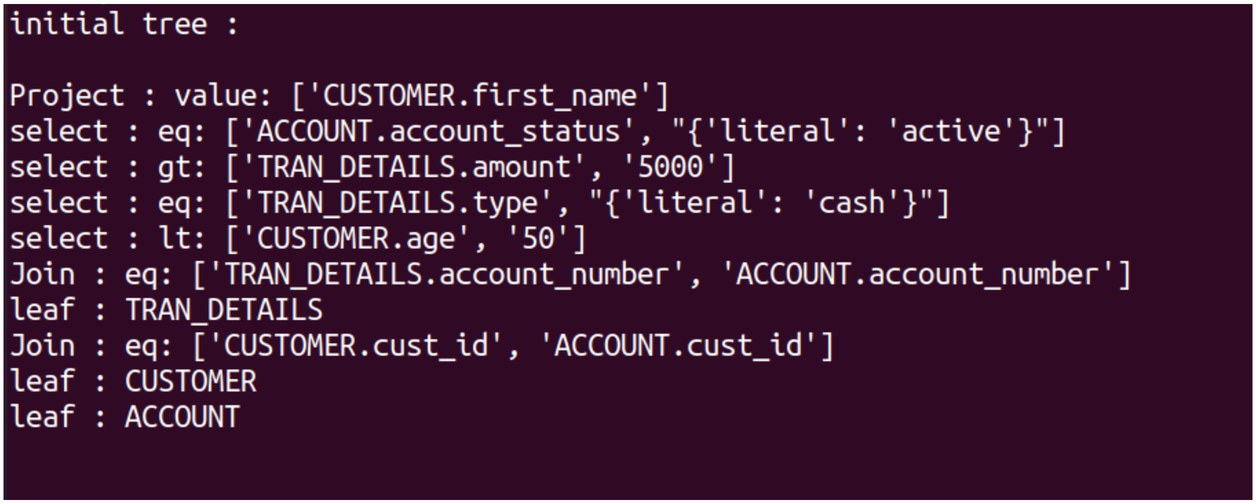
Example Queries :

Query1 :

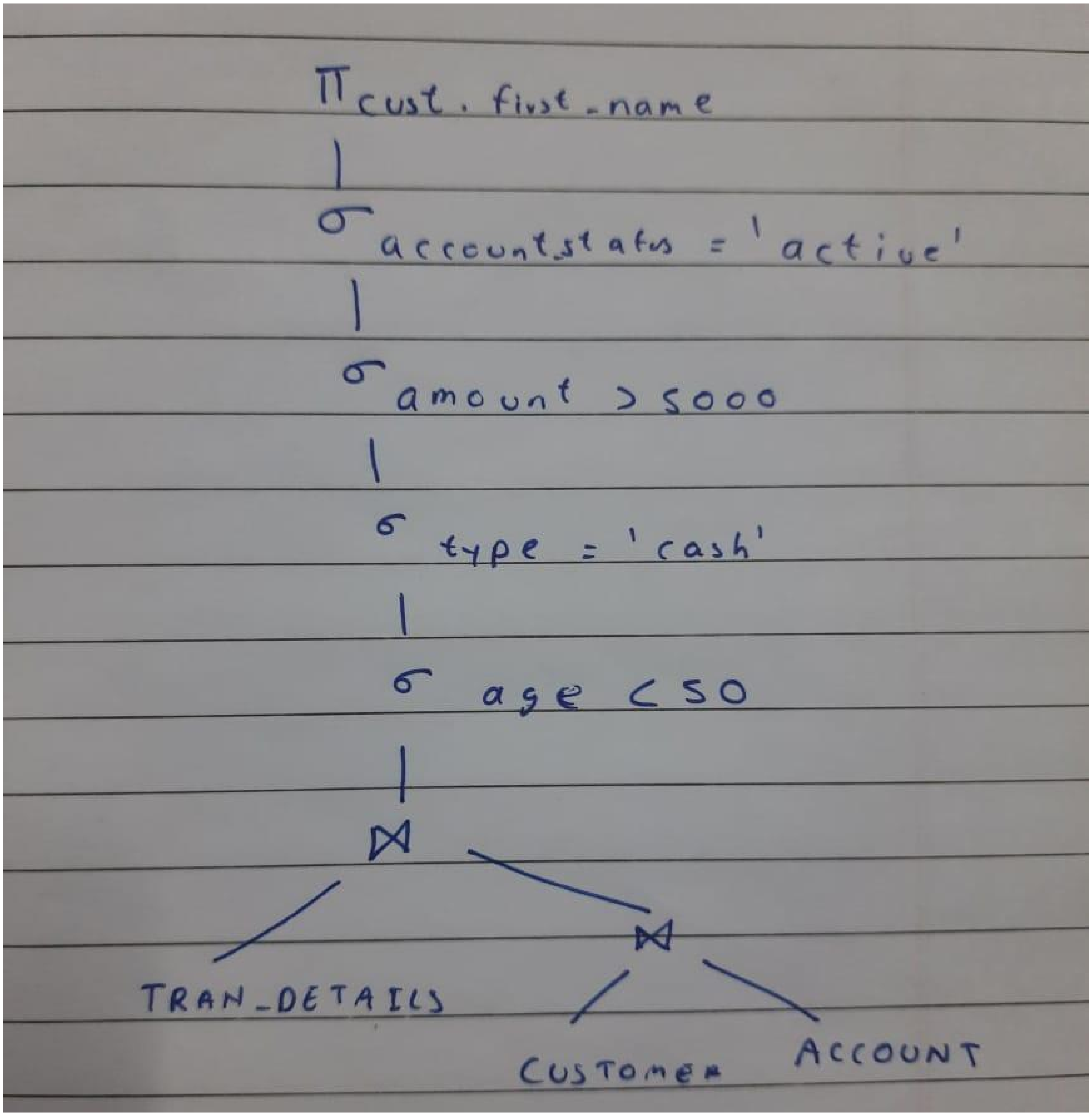


```
sql_statement = """
select CUSTOMER.first_name from CUSTOMER,ACCOUNT,TRAN_DETAILS
WHERE CUSTOMER.cust_id = ACCOUNT.cust_id
AND TRAN_DETAILS.account_number = ACCOUNT.account_number
AND ACCOUNT.account_status = 'active'
AND TRAN_DETAILS.amount > 5000
AND TRAN_DETAILS.type = 'cash'
AND CUSTOMER.age < 50
"""
```

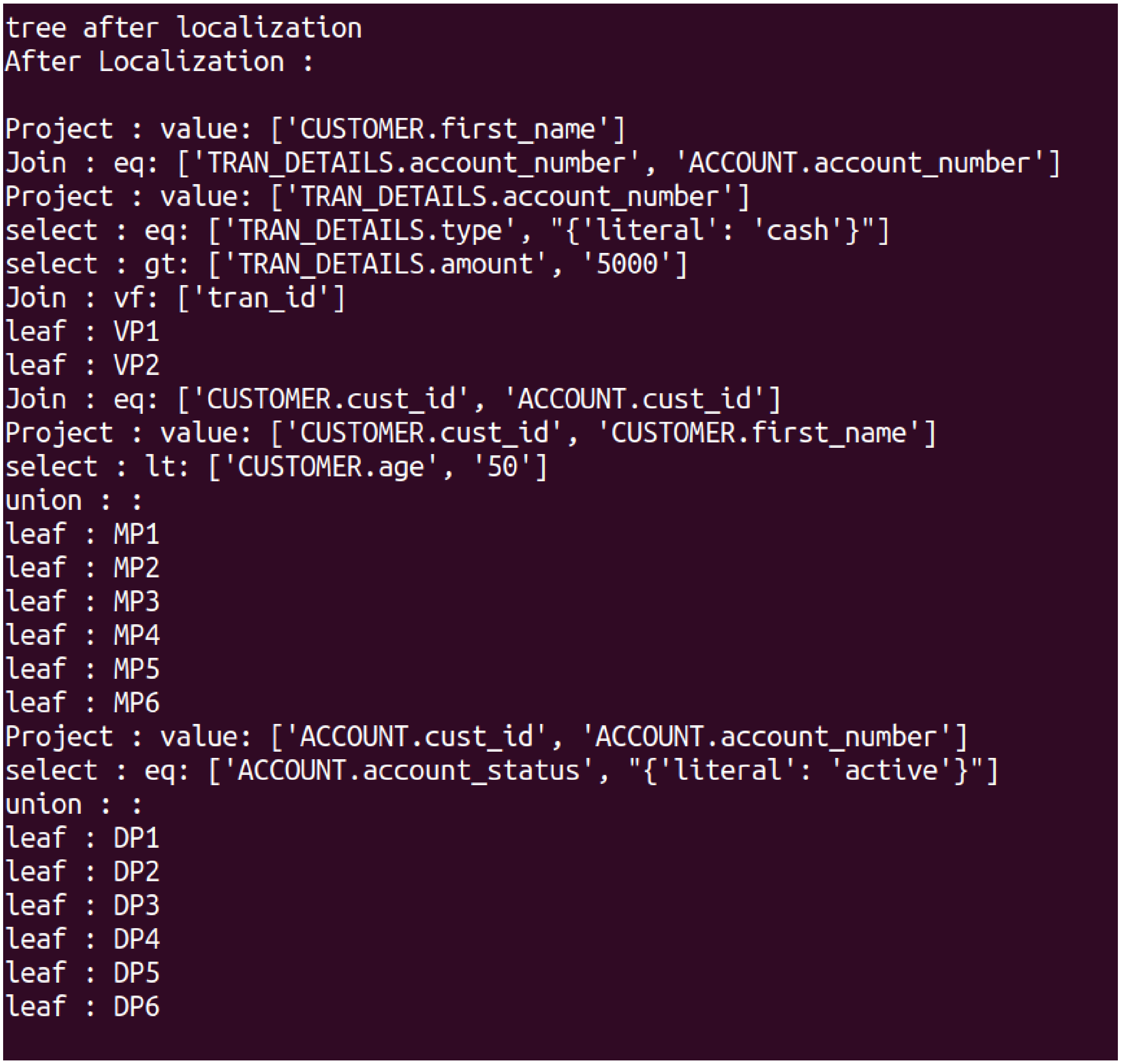
**Initial tree built : (Inorder traversal of tree)**



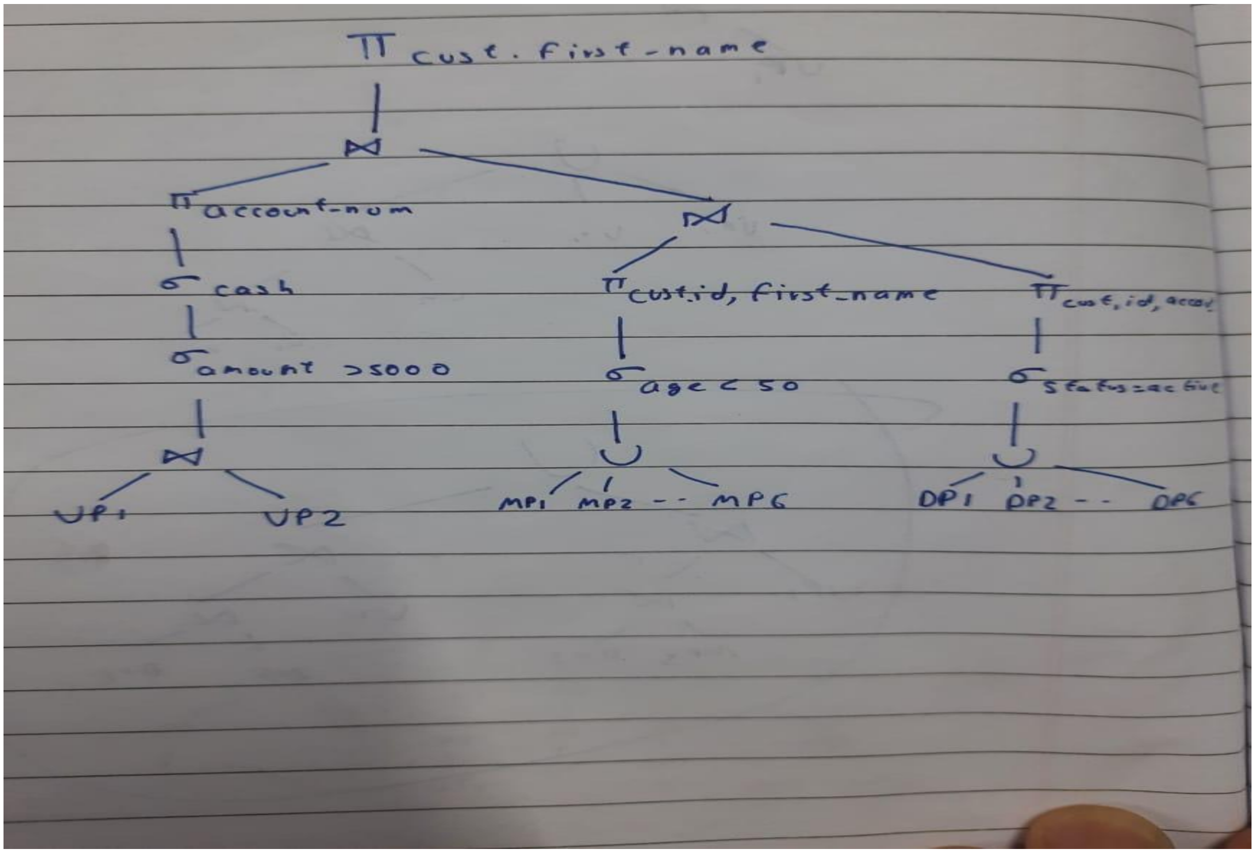
```
initial tree :

Project : value: ['CUSTOMER.first_name']
select : eq: ['ACCOUNT.account_status', "{'literal': 'active'}"]
select : gt: ['TRAN_DETAILS.amount', '5000']
select : eq: ['TRAN_DETAILS.type', "{'literal': 'cash'}"]
select : lt: ['CUSTOMER.age', '50']
Join : eq: ['TRAN_DETAILS.account_number', 'ACCOUNT.account_number']
leaf : TRAN_DETAILS
Join : eq: ['CUSTOMER.cust_id', 'ACCOUNT.cust_id']
leaf : CUSTOMER
leaf : ACCOUNT
```



Tree after pushing select and project down and also replace rel with fragments.
Tran_details → VP1,VP2 (Vertical fragmentation)
Customer → MP1 to MP6 (Horizontal fragmentation)
Account → Derived HF (DHF based on Customer)

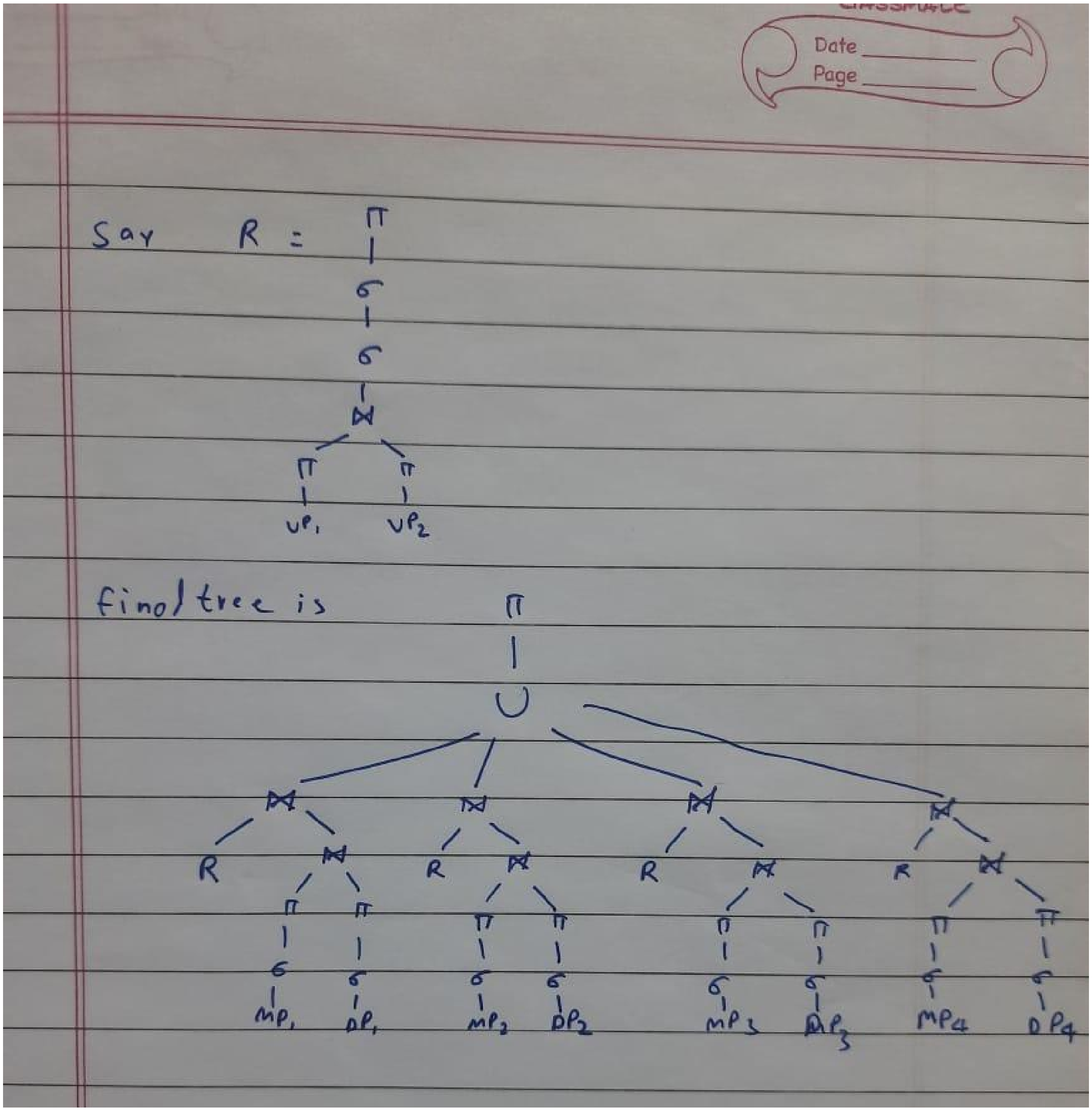


```
tree after localization
After Localization :

Project : value: ['CUSTOMER.first_name']
Join : eq: ['TRAN_DETAILS.account_number', 'ACCOUNT.account_number']
Project : value: ['TRAN_DETAILS.account_number']
select : eq: ['TRAN_DETAILS.type', "{'literal': 'cash'}"]
select : gt: ['TRAN_DETAILS.amount', '5000']
Join : vf: ['tran_id']
leaf : VP1
leaf : VP2
Join : eq: ['CUSTOMER.cust_id', 'ACCOUNT.cust_id']
Project : value: ['CUSTOMER.cust_id', 'CUSTOMER.first_name']
select : lt: ['CUSTOMER.age', '50']
union : :
leaf : MP1
leaf : MP2
leaf : MP3
leaf : MP4
leaf : MP5
leaf : MP6
Project : value: ['ACCOUNT.cust_id', 'ACCOUNT.account_number']
select : eq: ['ACCOUNT.account_status', "{'literal': 'active'}"]
union : :
leaf : DP1
leaf : DP2
leaf : DP3
leaf : DP4
leaf : DP5
leaf : DP6
```


π cust.first-name
⋈
π account-num
σ cash
σ amount >5000
⋈
UP1
UP2
⋈
π cust.id, first_name
σ age < 50
∪
MP1
MP2 - -
MP6
π cust.id, accou
σ status=active
∪
DP1
DP2 - -
DP6

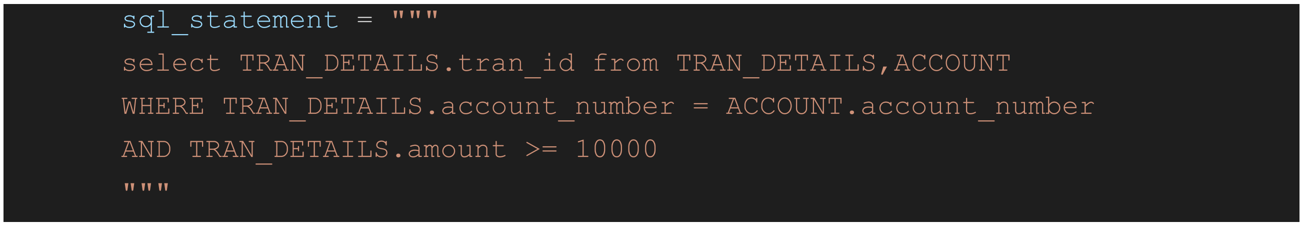
**Final Tree is :**

- MP5 and MP6 are reduced as they have age>50 as predicate
- The join between DPi and MPj where (i != j) are null. That is also reduced.
- Both the VP will be present and attributes from both fragments are needed.



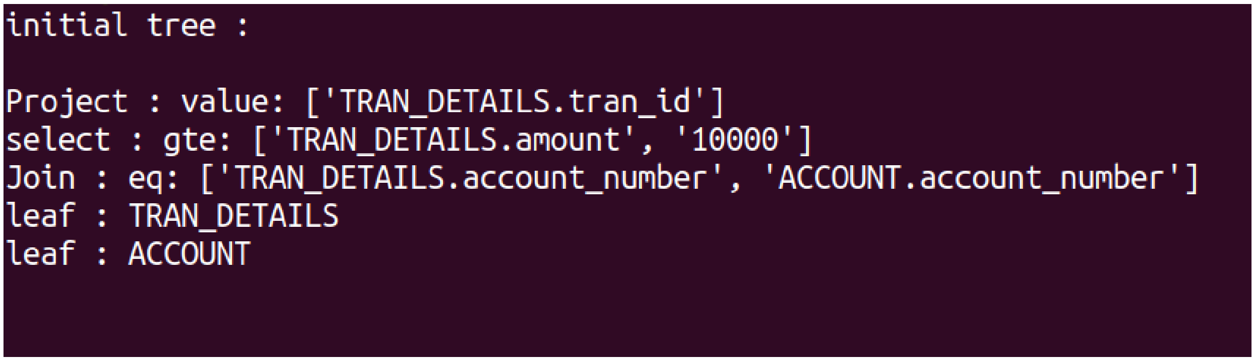
**Query 2 :**

We can see one of the vertical fragments not being used in this query.

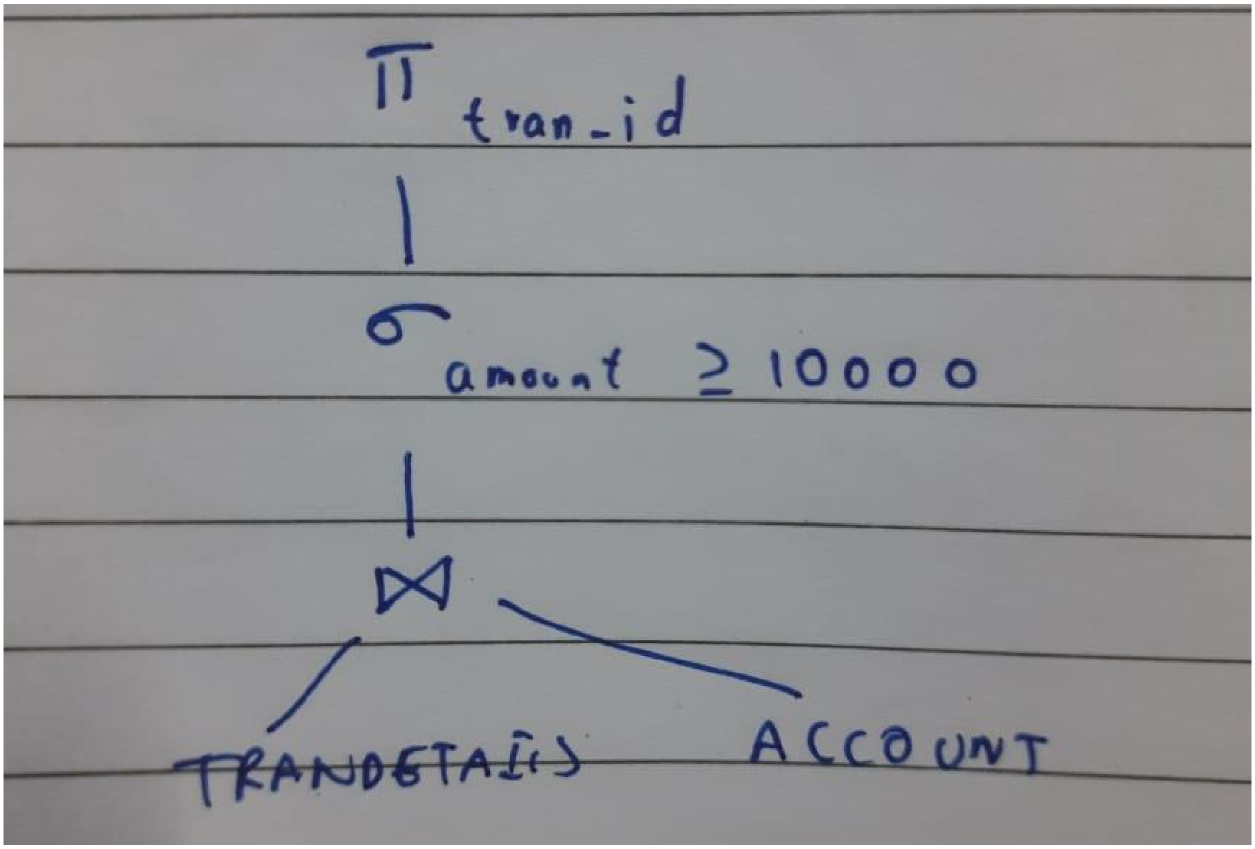


```
        sql_statement = """
        select TRAN_DETAILS.tran_id from TRAN_DETAILS,ACCOUNT
        WHERE TRAN_DETAILS.account_number = ACCOUNT.account_number
        AND TRAN_DETAILS.amount >= 10000
        """
```

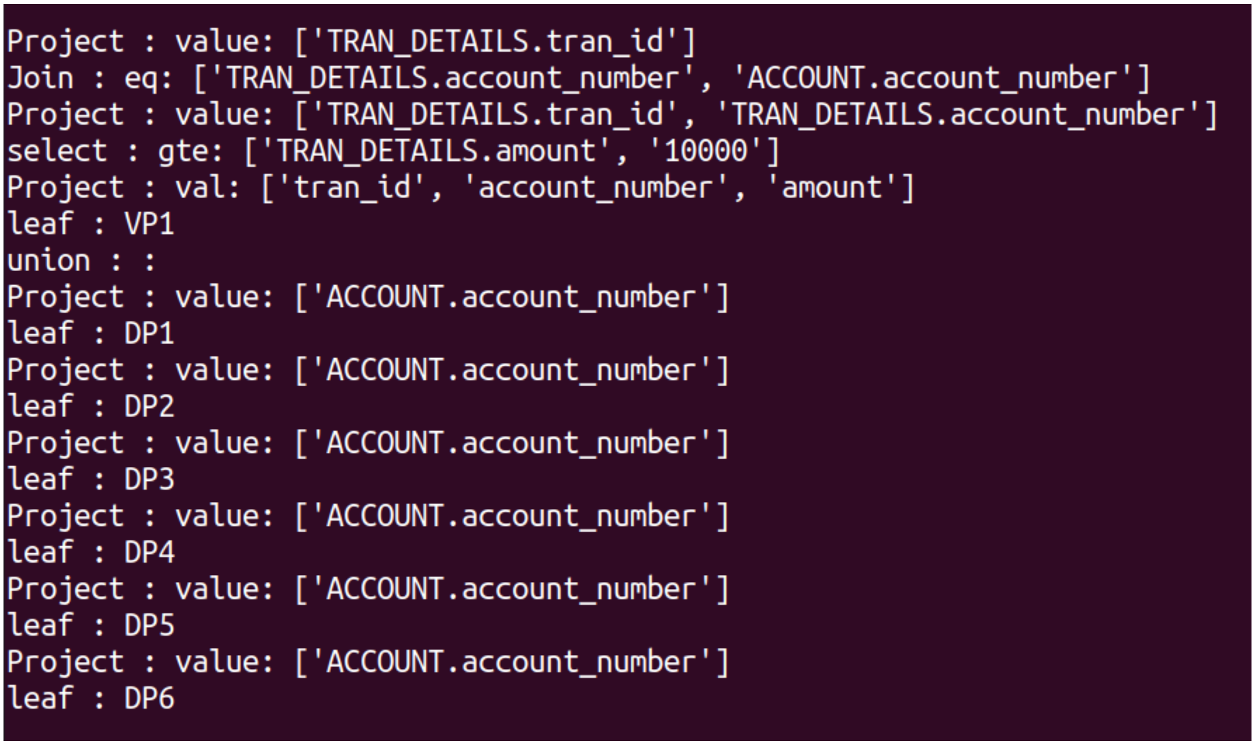
Initial tree :



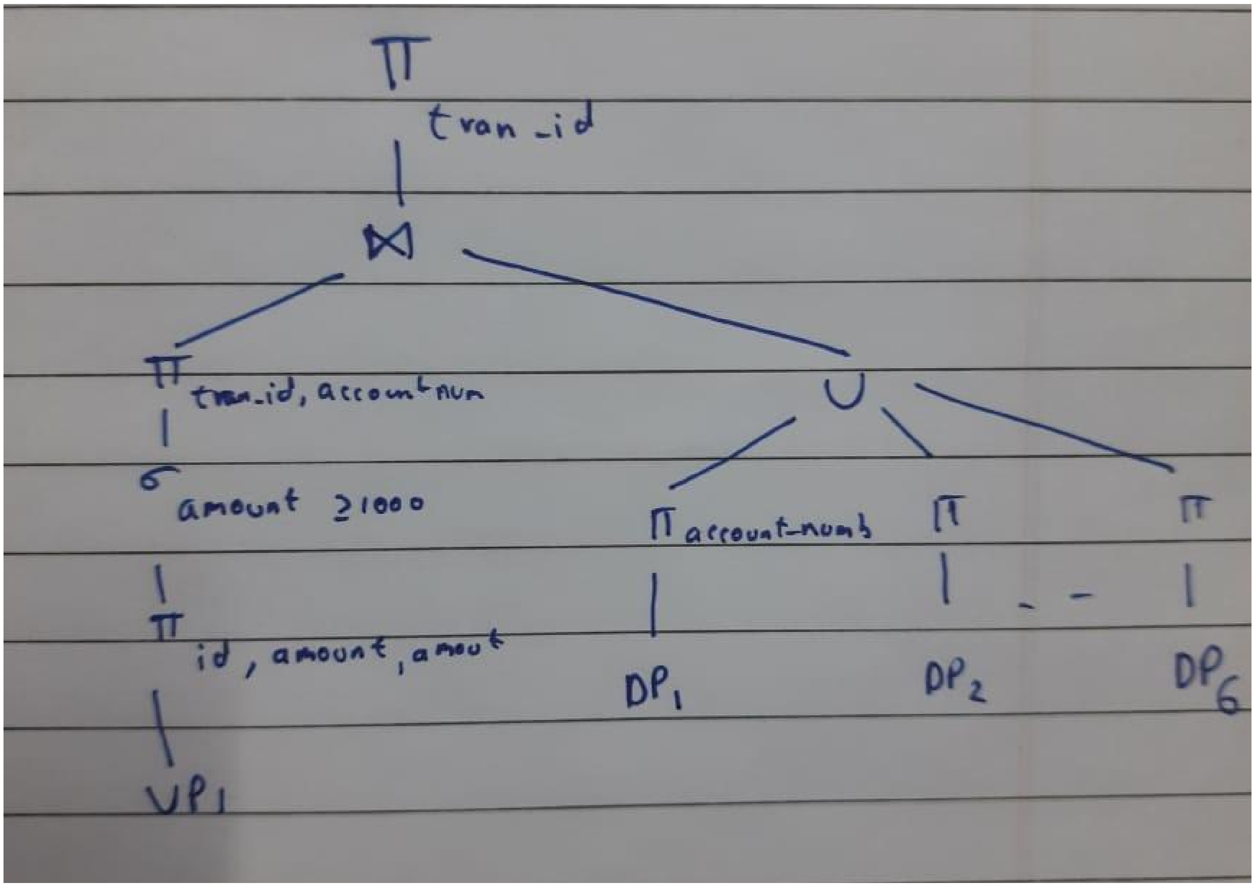
```
initial tree :

Project : value: ['TRAN_DETAILS.tran_id']
select : gte: ['TRAN_DETAILS.amount', '10000']
Join : eq: ['TRAN_DETAILS.account_number', 'ACCOUNT.account_number']
leaf : TRAN_DETAILS
leaf : ACCOUNT
```



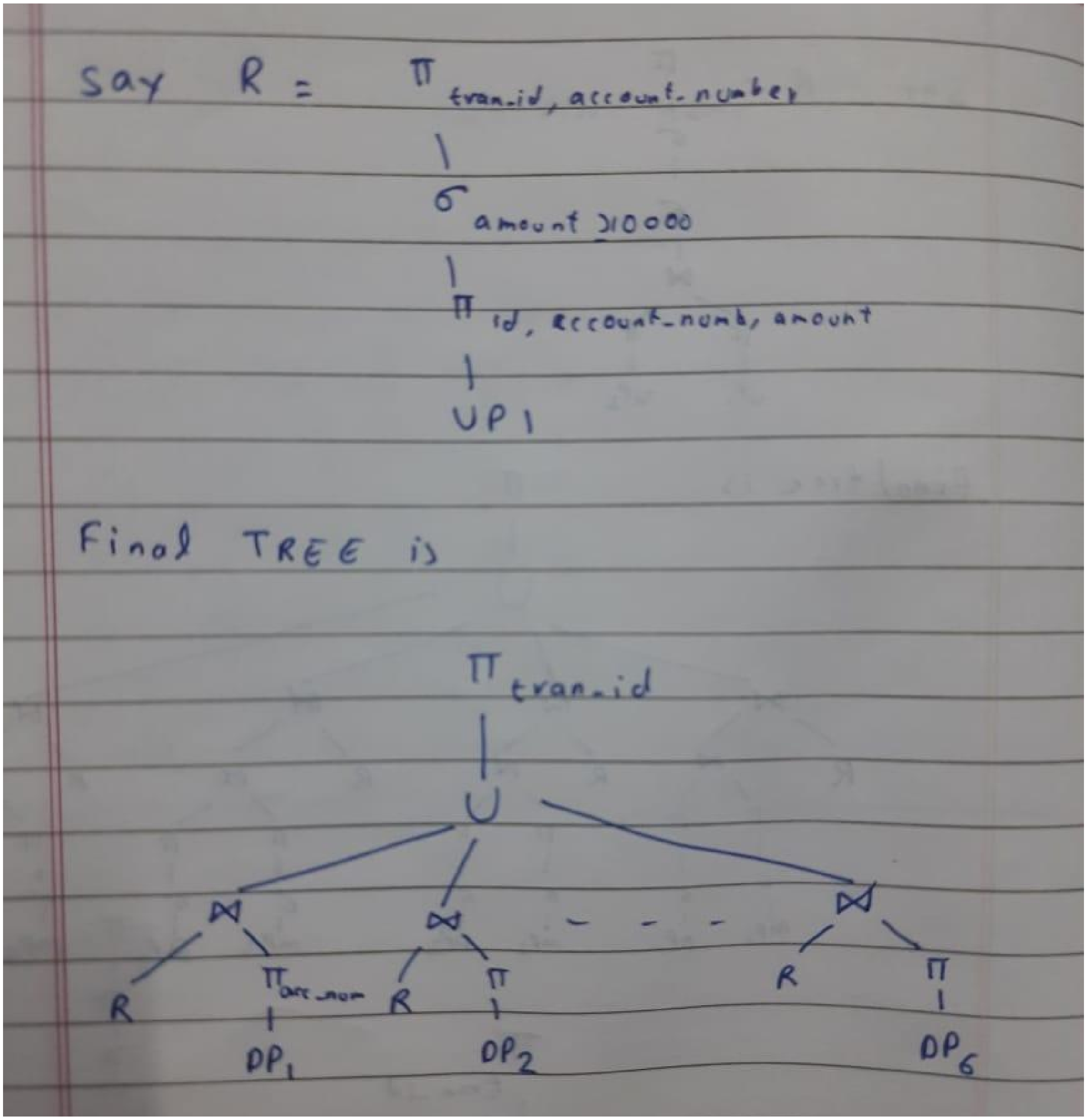
Localized tree output : (Before dist of join over union)



```
Project : value: ['TRAN_DETAILS.tran_id']
Join : eq: ['TRAN_DETAILS.account_number', 'ACCOUNT.account_number']
Project : value: ['TRAN_DETAILS.tran_id', 'TRAN_DETAILS.account_number']
select : gte: ['TRAN_DETAILS.amount', '10000']
Project : val: ['tran_id', 'account_number', 'amount']
leaf : VP1
union : :
Project : value: ['ACCOUNT.account_number']
leaf : DP1
Project : value: ['ACCOUNT.account_number']
leaf : DP2
Project : value: ['ACCOUNT.account_number']
leaf : DP3
Project : value: ['ACCOUNT.account_number']
leaf : DP4
Project : value: ['ACCOUNT.account_number']
leaf : DP5
Project : value: ['ACCOUNT.account_number']
leaf : DP6
```



Final Reduced Tree :



Say R =

Final TREE is