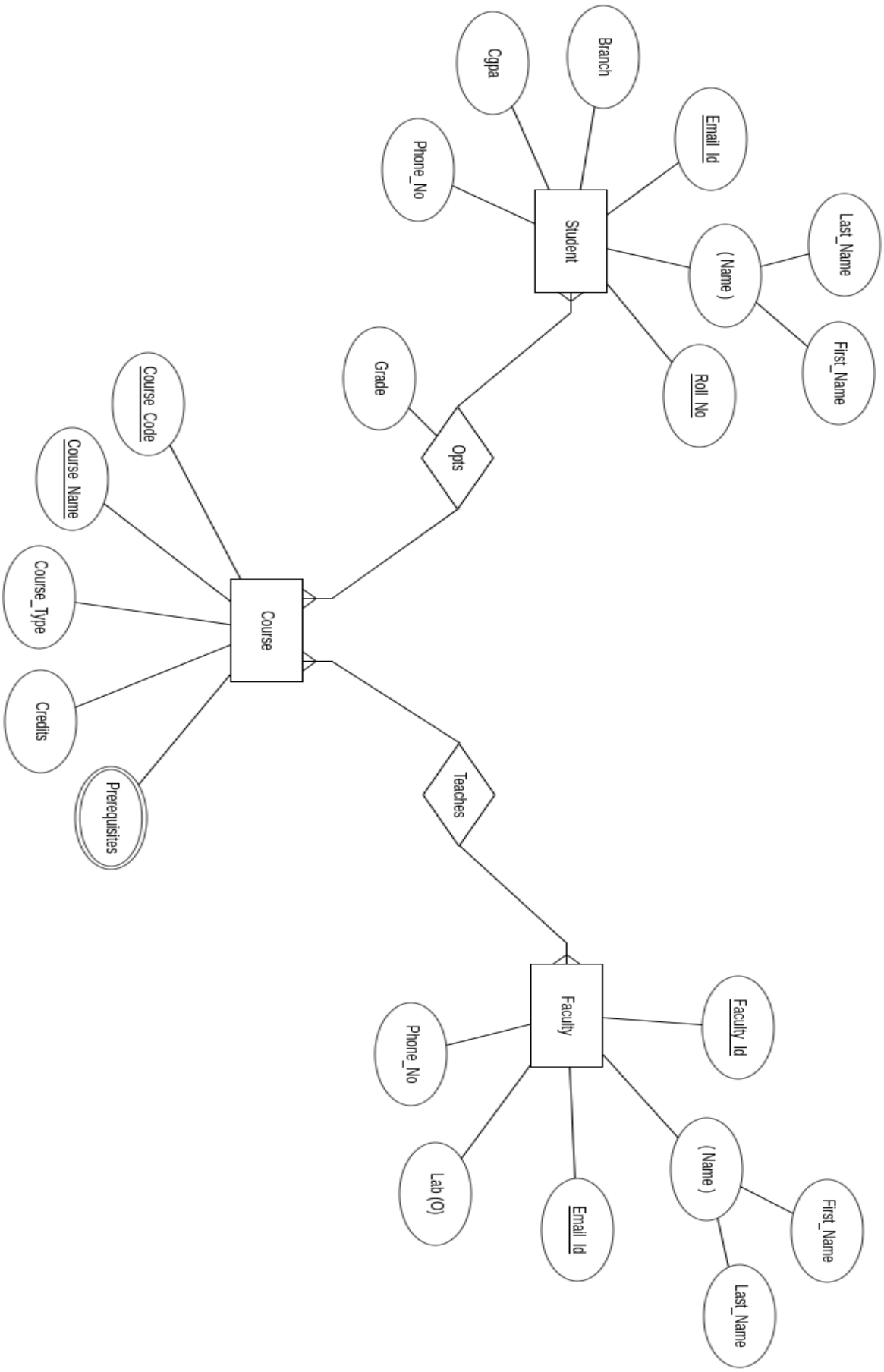
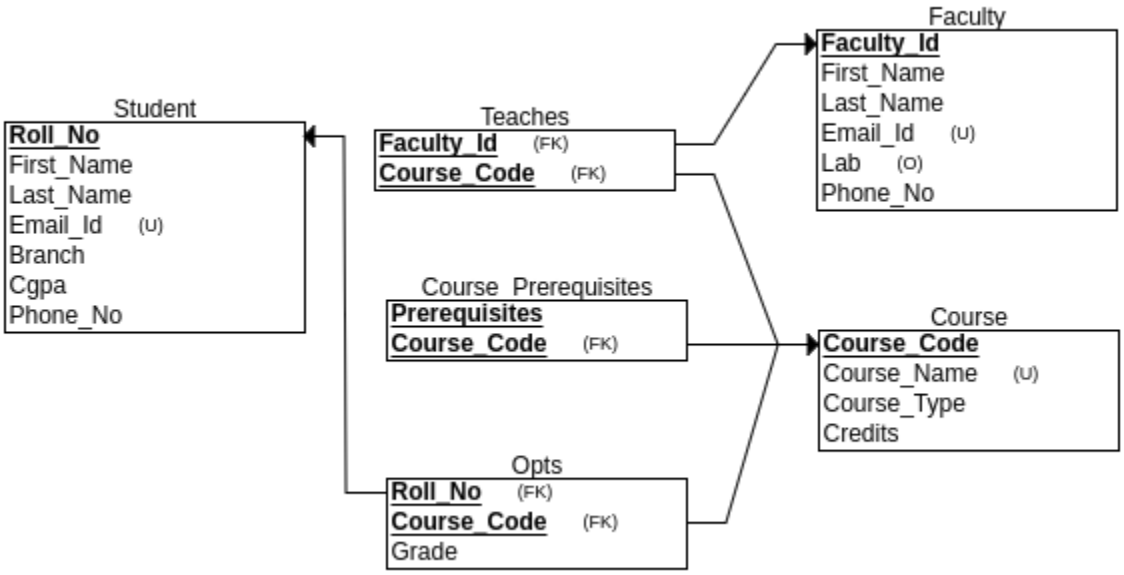
Distributed Data Systems
PROJECT PHASE-2
Report

Team - QuarantinedAgain
Team Members - Manas Kabre(2018111014)
- Nikunj Nawal(2018111011)

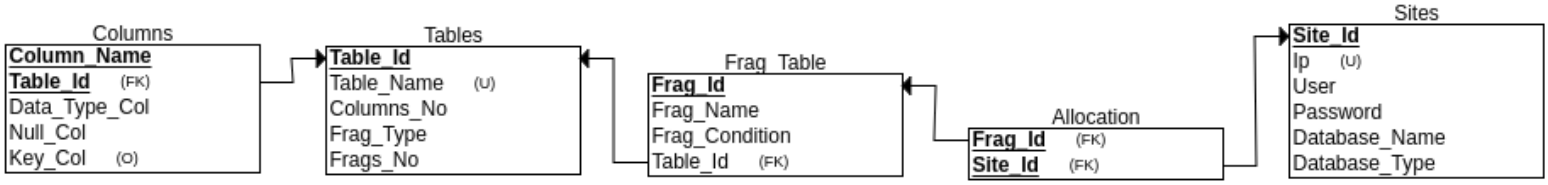
ER Diagram of The Application-



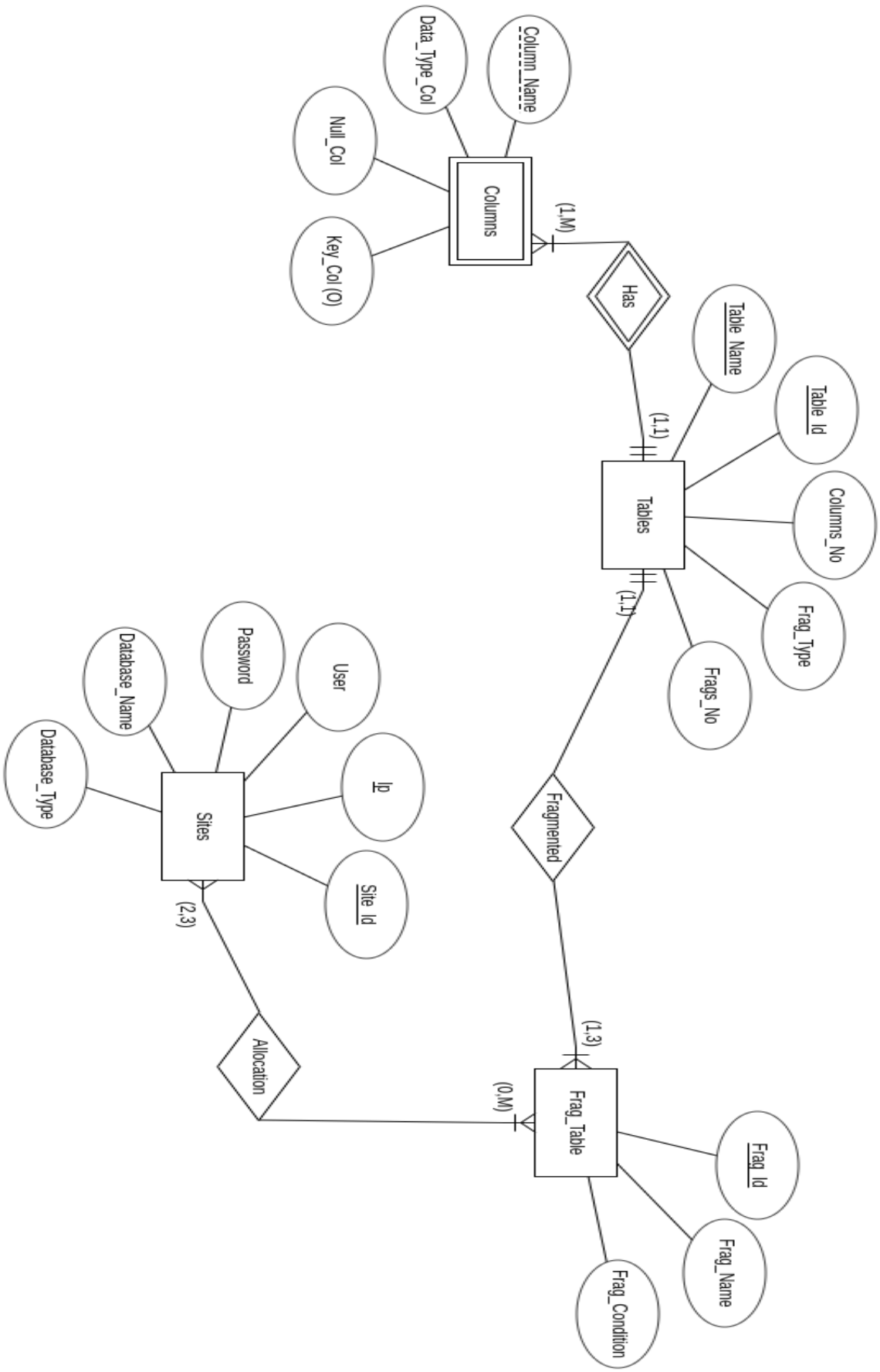
Relational Model of The Application-



Relational Model of The System Catalogue-



ER Diagram of the System Catalogue-



Explanation of the Code with example

Query :

Select reserve_id,name,city,price from Room, Guest, Reserve Where Room.reserve_id = Reserve.reserve_id and Room.reserve_id = Guest.reserve_id and Room.city = 'Mumbai' and Guest.guest_id < 20 and Room.reserve_id > 2 Group by name,price Having price > 3

```
PARSED QUERY :
SELECT reserve_id,
       name,
       city,
       price
FROM Room,
     Guest,
     Reserve
WHERE Room.reserve_id = Reserve.reserve_id
  AND Room.reserve_id = Guest.reserve_id
  AND Room.city = 'Mumbai'
  AND Guest.guest_id < 20
  AND Room.reserve_id > 2
GROUP BY name,
         price
HAVING price > 3
```

Query Tree :

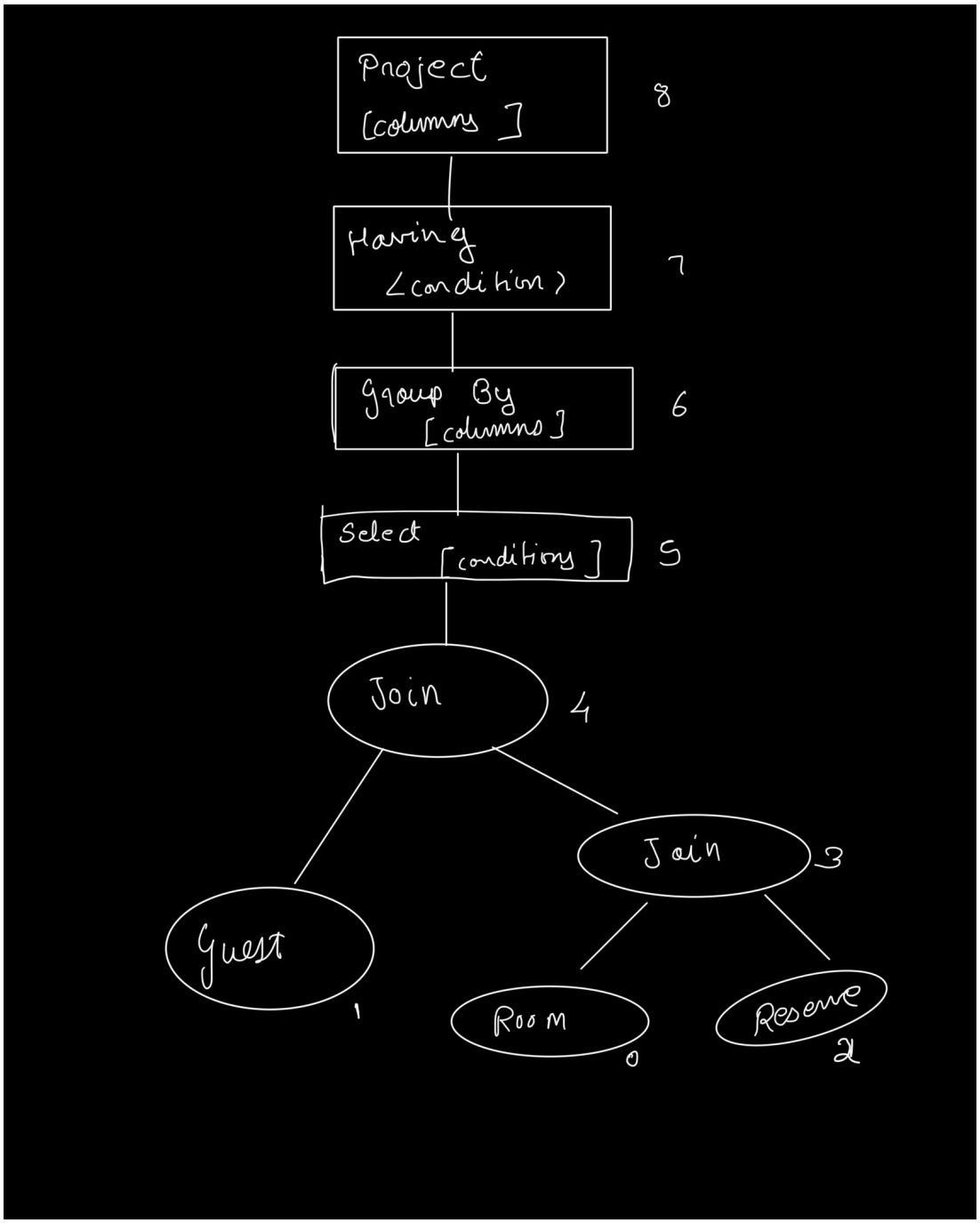
```
Decomposed Query Tree

0 -> {'Key': 'Table', 'Value': 'Room', 'Condition': []}
1 -> {'Key': 'Table', 'Value': 'Guest', 'Condition': []}
2 -> {'Key': 'Table', 'Value': 'Reserve', 'Condition': []}
3 -> {'Key': 'Join', 'Value': 'Room_Reserve', 'Condition': [['Room', 'reserve_id'], ['Reserve', 'reserve_id']]}
4 -> {'Key': 'Join', 'Value': 'Room_Guest', 'Condition': [['Room', 'reserve_id'], ['Guest', 'reserve_id']]}
5 -> {'Key': 'Select', 'Condition': ["Room.city = 'Mumbai'", 'Guest.guest_id < 20', 'Room.reserve_id > 2']}
6 -> {'Key': 'GROUP BY', 'Condition': ['name', 'price']}
7 -> {'Key': 'HAVING', 'Condition': ['price > 3']}
8 -> {'Key': 'Project', 'Condition': ['reserve_id', 'name', 'city', 'price']}
Edge List ->
[[3, 0], [3, 2], [4, 3], [4, 1], [4, 5], [5, 6], [6, 7], [7, 8]]
```

The approach used to build the tree was a bottom-up approach.
The initial parse query is taken to build the query tree. The following steps are involved :

- Create nodes for all the tables involved. i.e. One node for each table which will also be a leaf node
- Perform all the joins and connect them by valid edges and add these in the edge list
- Add the remaining select clauses in the where clause other than the join clause to a list and create a node to define the selection details
- Add any other conditions like having, group by, etc to the tree
- Finally add the Projection node which has the projected columns in the list of the condition Key.

The query example taken is a huge query involving almost every possible scenario. The following is the visualised version of the printed tree :



The indexing mentioned in the diagram as followed by the edge_list.

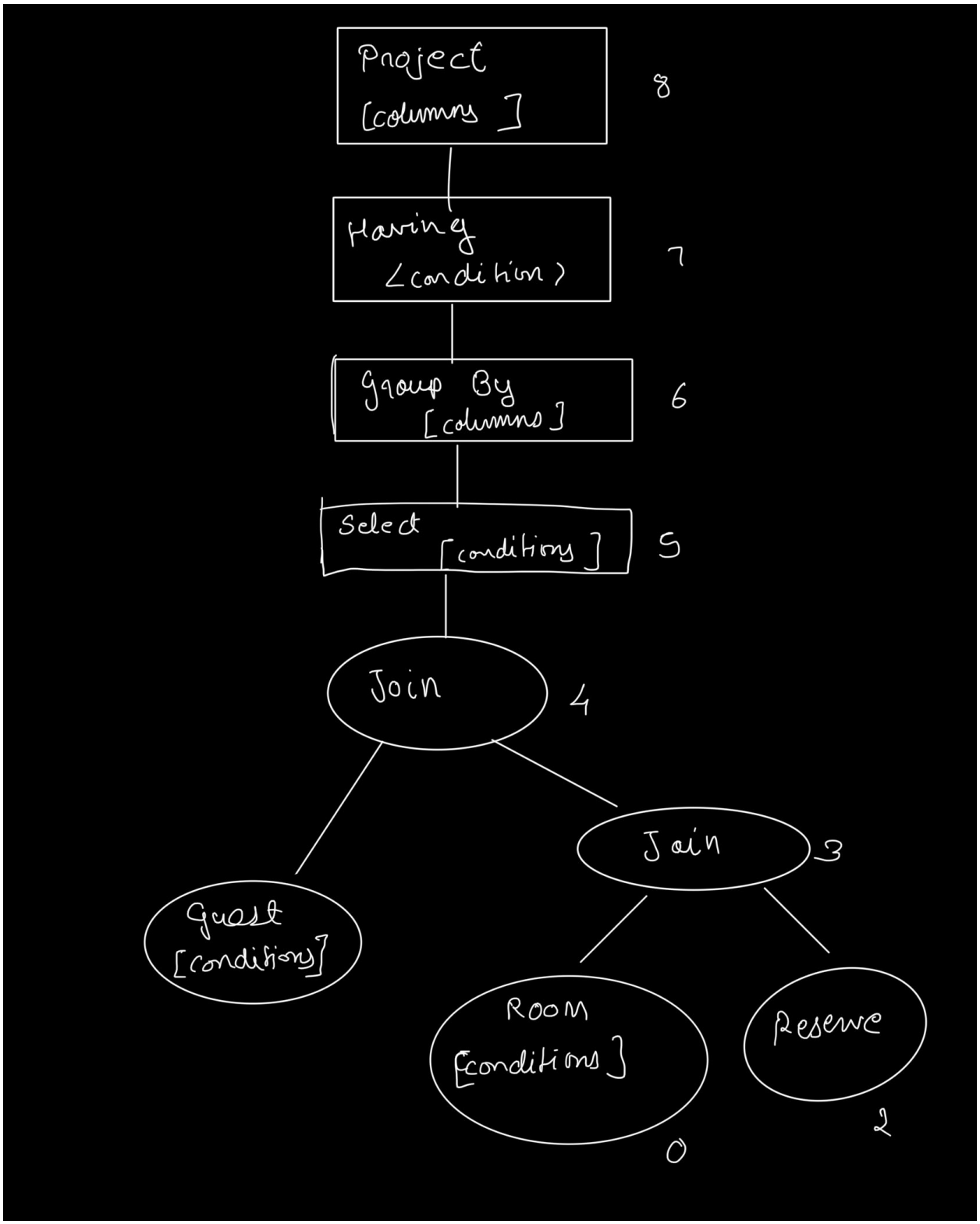
Query_Tree with Heuristic optimisation following transformation rules-

```
Decomposed Query Tree with Heuristic optimisations

0 -> {'Key': 'Table', 'Value': 'Room', 'Condition': ["Room.city = 'Mumbai'", 'Room.reserve_id > 2']}
1 -> {'Key': 'Table', 'Value': 'Guest', 'Condition': ['Guest.guest_id < 20']}
2 -> {'Key': 'Table', 'Value': 'Reserve', 'Condition': []}
3 -> {'Key': 'Join', 'Value': 'Room_Reserve', 'Condition': [['Room', 'reserve_id'], ['Reserve', 'reserve_id']]}
4 -> {'Key': 'Join', 'Value': 'Room_Guest', 'Condition': [['Room', 'reserve_id'], ['Guest', 'reserve_id']]}
5 -> {'Key': 'Select', 'Condition': []}
6 -> {'Key': 'GROUP BY', 'Condition': ['name', 'price']}
7 -> {'Key': 'HAVING', 'Condition': ['price > 3']}
8 -> {'Key': 'Project', 'Condition': ['reserve_id', 'name', 'city', 'price']}
Edge List ->
[[3, 0], [3, 2], [4, 3], [4, 1], [4, 5], [5, 6], [6, 7], [7, 8]]
```

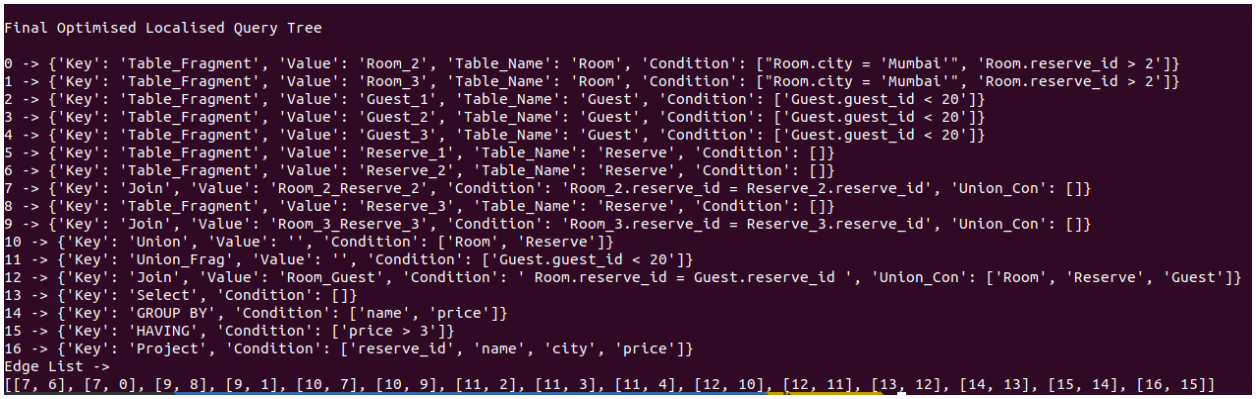
The following steps were involved :

- Note that the initial query tree made was partially optimised as it involved no cartesian product and other things like that
- The Select statements are pushed down the tree in this optimization so as to have smaller joins.
- Below is the visual representation of the tree from this part, which is similar to the previous tree but has certain conditions in the leaf nodes which are the conditions to apply on each table before performing any further operations.



The indexing mentioned in the diagram as followed by the edge_list.

Final Localised and Optimised Tree :

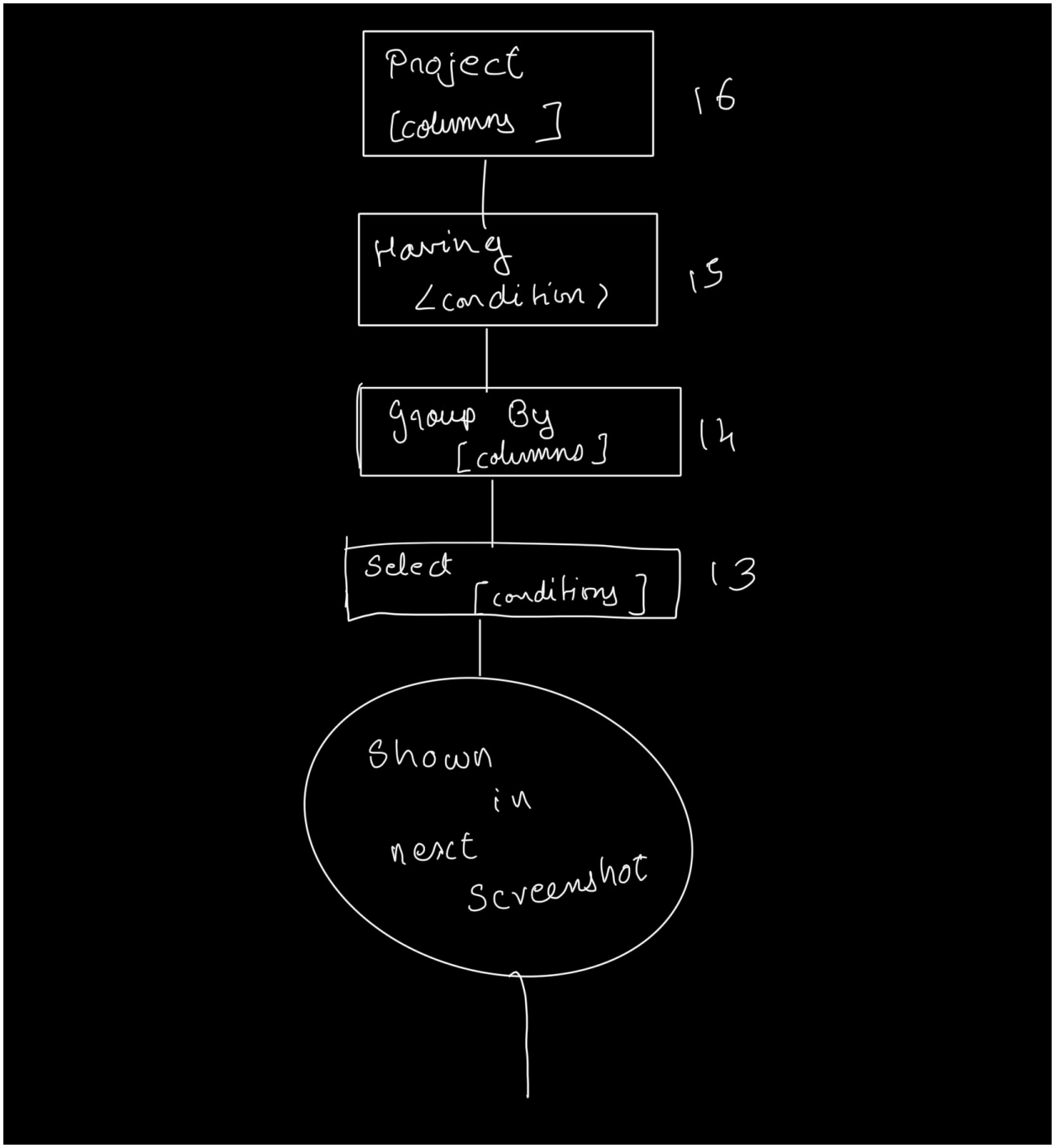


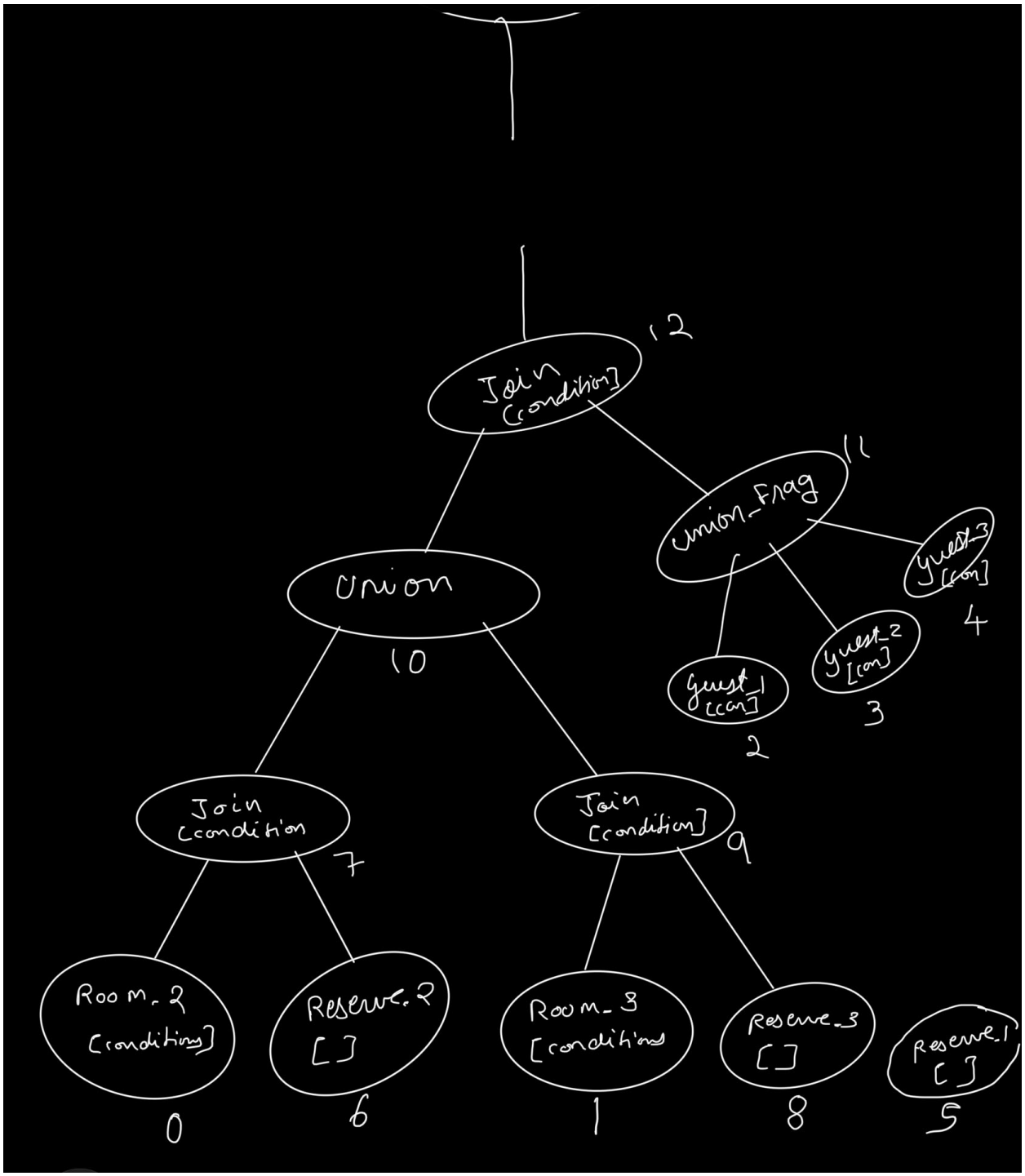
This is the final step of the algorithm. In this a new tree is created again using the bottom up approach, with the help of the available query tree. The following are the steps :

- As was done in the creation of the query tree, even for the localised query tree leaf nodes are created for each of the valid fragments of HF and VF. Note that DHF fragments are not taken into account at this step. By Valid fragments we refer to those fragments who have contribution of at least one record towards the final output. This is checked with the help of the fragmentation condition available in the system catalogue and the conditions which come in the query. If the intersection of the query conditions with the fragmentation conditions is null => that fragment has no contribution and hence will not be considered.
- The next step is to add the DHF fragments and their join with Parent HF(if any). All the DHF fragments are added to the nodes and if their Parent HF exist, then the corresponding join is performed and if it does not exist, then the node is left without any edge and would never come into our further calculations (like Reserve_1 in the example we have taken). The DHF joins are performed first as that is the most optimal approach to go about with the joins. These joins create new nodes and a union node is created by merging the DHF fragments joins for a given table.
- The next step is to add the other joins. This is done by iterating over the conditions in the where clause and checking if there is any join condition left. If there is any, then it is accordingly added to the tree

by following an approach which checks if there are any joins already performed on the participating table, and if there are then those joins are used to create the new join. If not then all the fragments of that table are merged into a union_frag node and then this node is used to perform the join.

- Finally the remaining select statements, having, group by project, etc are added to complete the tree.
- Below is the image of the visualisation of the tree.





The indexing mentioned in the diagram as followed by the edge_list.

Please note the node numbered 5, this node is just there without any edge. This node is added into the tree to maintain completeness. It is a node which belongs to the DHF table. If there is a query which involves the DHF table without its parent HF table then the DHF nodes need to be added in the tree's node list and hence the extra addition in this case although it is not required. However while executing the tree we would use some kind of

DFS or BFS approach from the Root node which would imply no participation of this extra node maintaining our optimised approach.

Note :

In case there is any misunderstanding in the explanation, we are willing to explain the same over a teams call, since the procedure followed is very tough to explain in the form of a report. However, we have tried our best to explain the same as appropriately as possible.