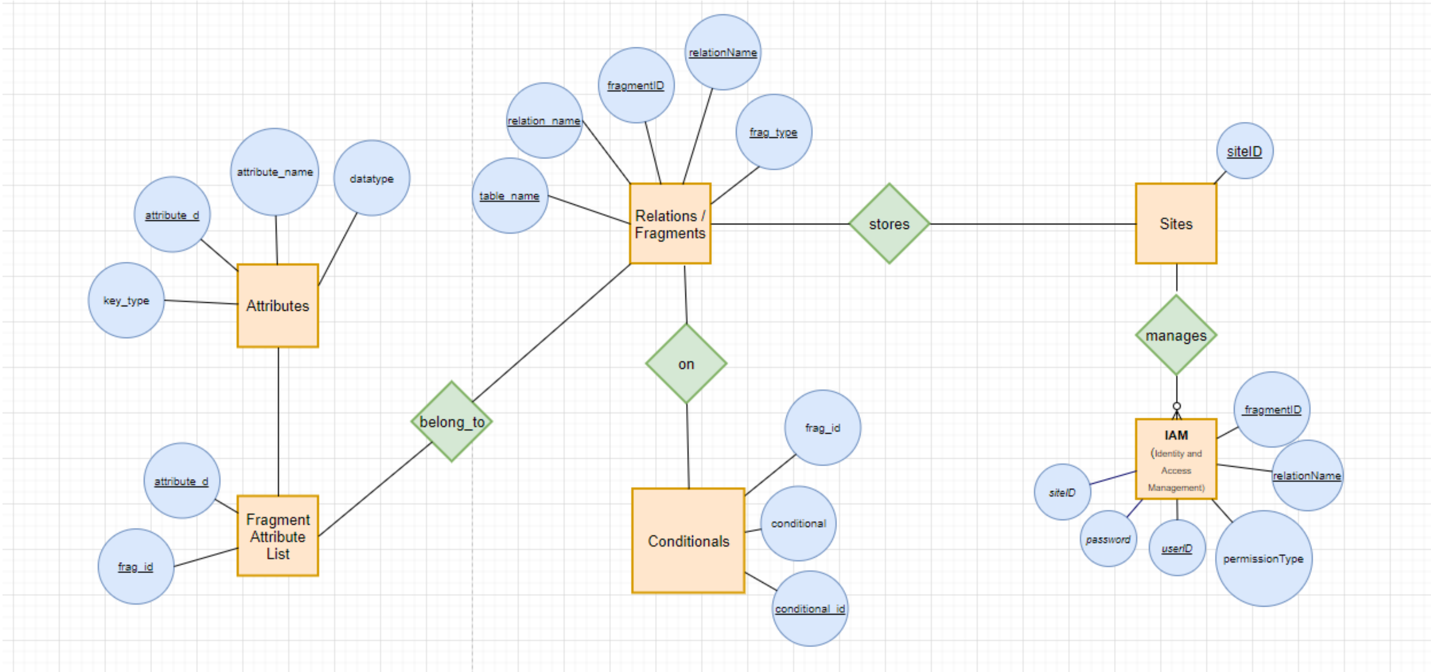
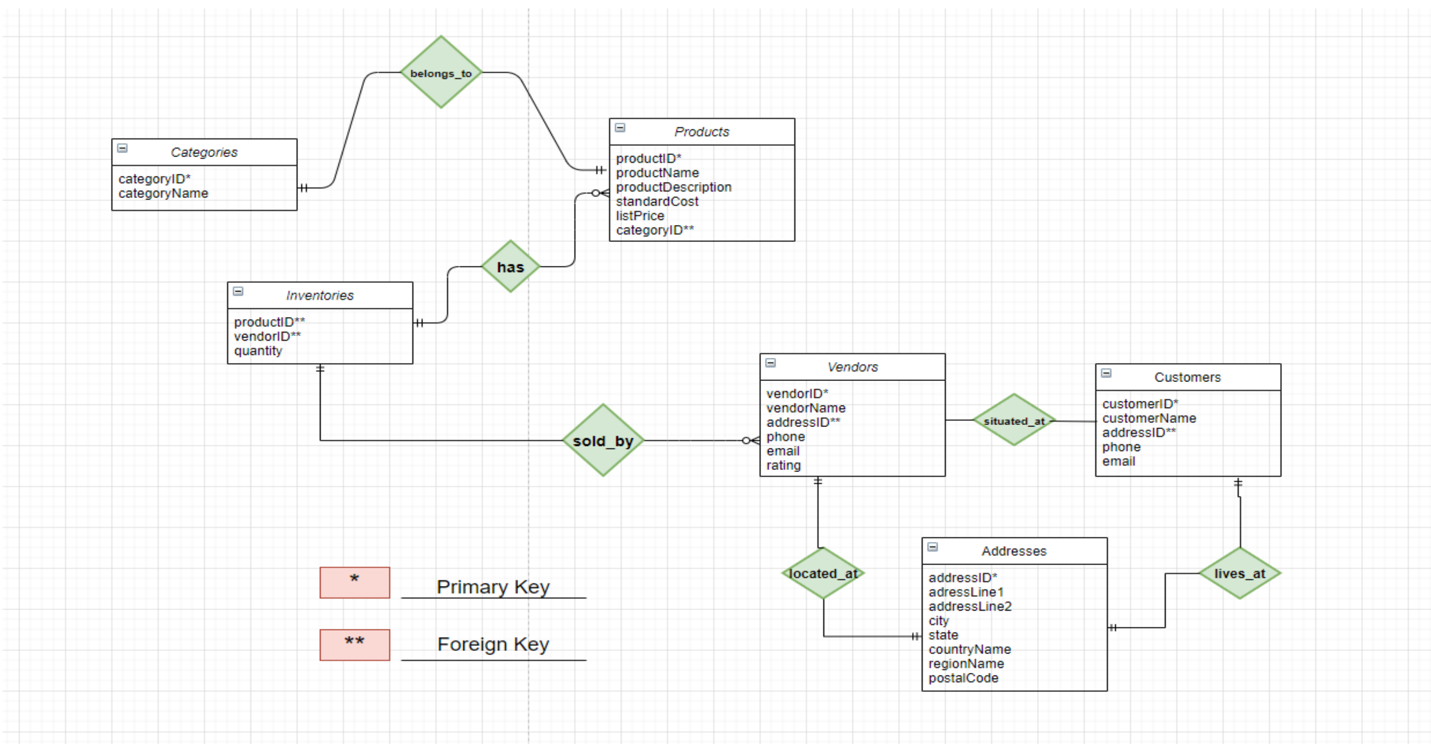
## SYSTEM CATALOG ER

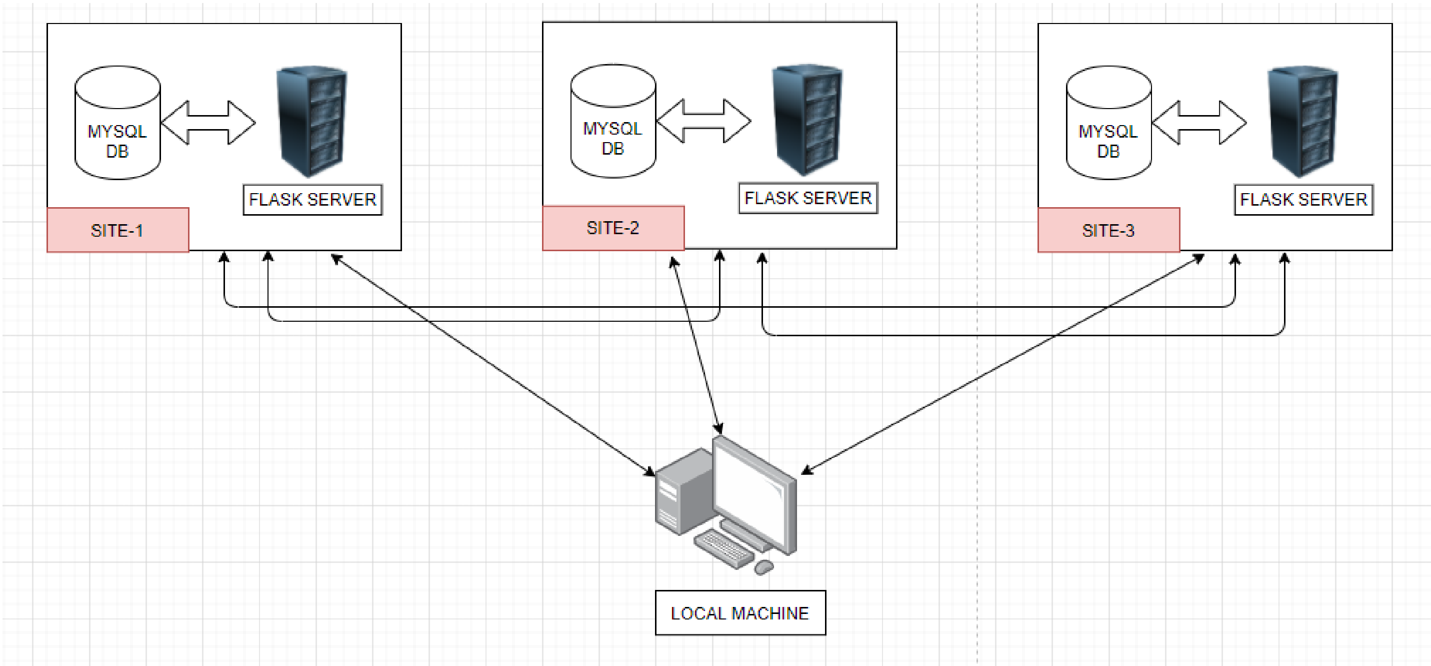



## APPLICTION DB ER




## SYSTEM DESIGN

The main script is run from a local machine that interacts with the 3 sites where data is stored. At each of the 3 sites, a flask application is running that listens to different types of queries and executes them and return the result to the callee.




### FLASK SERVER

There are 3 types of end points for each flask server :

1. Execute read-only query
2. Transfer table to another site
3. Move R_prime = (R semi_join S) to site of S

### MYSQL DB

The MYSQL DB contains the fragmented data and system catalog tables.

## Fragmentation / Allocation of Data

The complete data (tables/system catalog tables) are stored at Site-1 initially. The fragmentation and allocation schema can be provided as input (in specified format) to the script 'csv_parser.py' , which fragments and allocates tables as specified.
The csv_parser.py exists at SITE-1, with initial system catalog tables.

## Query Execution

STEP-1 Query Localization
Based on the fragmentation schema the query , a localized query is generated. We have used a bottom up approach, where we push down the union and bring up the join in the initial query tree, and exhaustively list all the possible joins.
STEP-2 Vertical Fragmentation based pruning
If the attributes specified in the query are not preent in the fragment, then the join is not included in the final query result.
STEP-3 Predicate based pruning
The next step involves pruning the listed joins based on the fragmentation conditions and specified where clause conditions.

DATA STRUCTURE USED -> TREE

**CHECKPOINT-1 LOCALIZED QUERIES READY**

STEP-4 Query Optimization
The generated joins are reordered based to get the optimal join order based on the fragment sizes and allocation schema. This optimization is based on SDD-1 and move small, heuristics based optimization strategies. We enumerate all possible join orders and find the cost of each order and select the one with minimal cost. Complexity : O(n!) where n is the number of joins in the join order [ n! -> all permutations of joins ]

**CHECKPOINT-2 Optimized Localized Query (Heuristics Based)**

STEP- 5 Generating Infix Query Expression
The query tree built in before stages is converted into an infix query expression. Then we use stack to solve the infix query expression. The intermediate results are stored in temporary tables.
There are 2 set if operations, 'JOIN' and 'UNION'.
'JOIN' -> The where clauses are not applied/included in 'JOIN' generation.
'UNION' -> The where clauses are applied while running this query.

This step involves transferring data across sites and performing joins and unoins.

DATA STRUCTURE USED -> STACK

**CHECKPOINT-3 QUERY result ready**

## Aggregate Queries

The ddbms built, supports SUM, MIN, MAX and AVG aggregate functions. The 'SUM', 'MIN', 'MAX' are straightforward to compute from the sub queries. For computing AVG we have used a proxy function :-

AVG(attr) = SUM(attr) / SUM(COUNT(*))

**NOTE** -> THE SYSTEM STAYS IN SQL ENVIRONMENT ALL THE TIME WHILE EXECUTING QUERIES, NO QUERY IS EXECUTED BY PROCESSING THE TABLES IN MAIN MEMORY

## How to use?

QUERY ->
STEP-1 python3 main.py
STEP-2 input query and press enter
CSV PARSER
python3 csv_parser.py sys_cat.csv

## QUERIES

### Horizontal Fragmentation

1. select * from Products where Products.listPrice>300
2. select * from Products where Products.listPrice>=300 and Products.listPrice<=50000

### Vertical Fragmentation

1. select Vendors.vendorName from Vendors
2. select Vendors.vendorName from Vendors where Vendors.addressID=1
3. select * from Vendors where Vendors.vendorName='abc' or Vendors.vendorName='pqr' or Vendors.addressID=4 or Vendors.vendorName='pqrs'

### Horizontal + Derived Horizontal Join

1. select Products.productName from Products,Categories where Products.categoryID=Categories.categoryID

### Horizontal +Derived Horizonotal + No Fragmentation JOIN

1. select Products.productName from Products,Categories,Inventories where Products.categoryID=Categories.categoryID and Products.productID=Inventories.productID and Categories.categoryName="Fashion" and Products.listPrice>3000 and Products.listPrice<=50000

### Vertical Fragmentation + Derived Horizontal Fragmentation JOIN

1. select * from Inventories,Vendors where Vendors.vendorID=Inventories.vendorID

### Complex AND/OR clause JOIN

1. select Products.productName from Products,Categories where Products.categoryID=Categories.categoryID and (Products.listPrice>10000 or Products.listPrice<=500)

### Aggregate Queries

1. select categoryID, MAX(listPrice) from Products GROUP BY categoryID
2. 4. select SUM(listPrice) from Products GROUP BY categoryID
3. select MAX(listPrice) from Products where categoryID=1 GROUP BY categoryID
4. select Products.categoryID, AVG(listPrice) from Products, Categories where Products.categoryID = Categories.categoryID GROUP BY categoryID
5. select min(Products.listPrice) from Products GROUP BY Products.productName HAVING count(*)=1