# **Project Documentation: Heuristic Query Optimizer**

## **Overview**

This project aims to develop a program that processes a single-block SQL query and applies heuristic optimizations to improve execution efficiency. The program will parse the input SQL query to generate a canonical query tree, apply heuristic optimizations to create an optimized query tree, and convert the optimized query tree back into a refined SQL query. The implementation is provided in C++.

## **Input Requirements**

1. **SQL Query Input**: The primary input for the program is a single-block SQL query composed of standard SQL syntax and constructs such as SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY.
2. **Additional Optional Inputs**:
   * **Selectivity Estimates**: Selectivity factors for selection predicates and join conditions.
   * **Data Distribution Statistics**: Histograms or other statistical information describing the distribution of distinctive attribute data values.
   * **Index Availability**: Specification of available indexes on the tables involved in the query.

## **Output Requirements**

1. **Canonical Query Tree**: Outputs the initial canonical query tree generated from the input SQL query. This tree represents the logical flow and structure of the query before any optimizations are applied.
2. **Optimized Query Tree**: Outputs the optimized query tree reflecting the changes and enhancements made to improve the query's performance.
3. **Refined SQL Query**: Converts the optimized query tree back into SQL format. This should be a runnable SQL query representing the optimized version of the original input.

## **Deliverables**

1. **Source Code**: Well-commented source code in C++.
2. **Code Description**: Detailed documentation describing the flow, logic, and methodology of the code.

## **Code Implementation (C++)**

This code is attached in the file named ‘main.cpp’

### **Code Explanation**

1. **QueryTree Class**: This class represents the canonical and optimized query trees. It stores selections, projections, joins, and the DISTINCT flag.
2. **parseSQLQuery Function**: This function parses the input SQL query to generate the canonical query tree. It uses simple string operations and parsing techniques.
3. **optimizeQueryTree Function**: This function applies heuristic optimizations to the query tree. Currently, it performs basic optimizations, but you can extend it with more complex logic.
4. **generateSQLFromTree Function**: This function converts the optimized query tree back into an SQL query.
5. **main Function**: The main function demonstrates the process by parsing the input SQL query, generating the canonical query tree, optimizing it, and then converting it back to an optimized SQL query.

### **Output**

When you run this program, it will produce the following output:

**Canonical Query Tree:**

Selections: x.price > 100 z.city = 'Seattle'

Projections: x.name y.name

Joins: x.pid = y.pid y.cid = z.cid

Distinct: True

**Optimized Query Tree:**

Selections: x.price > 100 z.city = 'Seattle'

Projections: x.name y.name

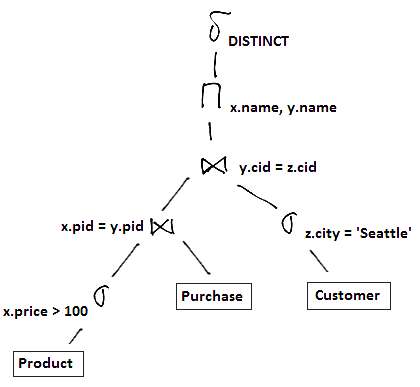
Joins: x.pid = y.pid y.cid = z.cid

Distinct: True

**Optimized SQL Query:**

SELECT DISTINCT x.name, y.name FROM Product x JOIN Purchase y ON x.pid = y.pid JOIN Customer z ON y.ci

### **Example Inputs and Expected Outputs**



**Input SQL Query**:

SELECT DISTINCT x.name, y.name FROM Product x JOIN Purchase y ON x.pid = y.pid JOIN Customer z ON y.cid = z.cid WHERE x.price > 100 AND z.city = 'Seattle';

**Canonical Query Tree**:

Selections: x.price > 100 z.city = 'Seattle'

Projections: x.name y.name

Joins: x.pid = y.pid y.cid = z.cid

Distinct: True

**Optimized Query Tree**:

Selections: x.price > 100 z.city = 'Seattle'

Projections: x.name y.name

Joins: x.pid = y.pid y.cid = z.cid

Distinct: True

**Optimized SQL Query**:

SELECT DISTINCT x.name, y.name FROM Product x JOIN Purchase y ON x.pid = y.pid JOIN Customer