Optimisation Strategy:

This optimiser class uses the **Heuristic method** to optimise SJDB queries. Each method in the class is designed to simplify and restructure the query plan to improve efficiency. **Comments have been added at important points in the code to make it easier to understand.** I used **4 main steps for optimisation**. The details are as follows:

1. Transform to Left-Deep Tree: *transformToLeftDeepTree(plan)*

Strategy: This optimization restructures the binary tree to a left-deep tree. The advantage of a left-deep tree is that it is aligned with the physical implementation of joins, which is prepared for subsequent heuristic optimisation. (Although QueryParser will parse the query statement into a left-deep tree, I added this method in consideration of passing non-left-deep queries directly to the optimiser which affects subsequent optimisation.)

Code Implementation: The method recursively navigates through the Product nodes of the query plan, ensuring that each Product node's right child is not another Product. If a right child is a Product, it restructures the tree to rotate the Product nodes, moving them to the left. This step readies the plan for efficient sequential joining of tables from left to right.

2. Pushdown Selections: *pushdownSelections(plan)*

Strategy: Selection pushdown is a critical optimization that moves selection operators (σ) closer to the leaf nodes of the tree. By applying filters as early as possible, it minimizes the number of tuples carried through the query plan, thus reducing I/O and computation.

Code Implementation: The pushdownSelections function employs a recursive approach to descend through the tree and push selection criteria down to the nearest possible Scan operations. When a selection predicate applies to the attributes of a subtree, it is pushed down to optimize data filtering at an early stage.

3. Combine Selection and Product to Join: *combineToJoin(plan)*

Strategy: The strategy identifies instances where a selection predicate following a Cartesian product can be converted into a join operation. Since join operations are more performant than the combination of a product and a subsequent selection, this restructuring benefits the overall query execution.

Code Implementation: The method detects patterns where a Select immediately follows a Product. It evaluates whether the selection predicate represents an

equijoin condition. If it does, it transforms these two operations into a single Join, which is typically more efficient in relational database systems.

4. Pushdown Projections: *pushdownProjects(plan)*

Strategy: The projection pushdown moves projection operators (π) down the query tree. The goal is to eliminate unnecessary columns from intermediate results as early as possible, reducing the volume of data processed and passed up the tree.

Code Implementation: Starting from the top-level projections, the method identifies the required attributes and pushes these projections down the tree. This method gradually filters out unnecessary attributes by recursion, thus avoiding unnecessary data loading and processing.

Appendix (Test Example):

Test SQL (Base on cat.txt):

SELECT projname

FROM Person, Department, Project

WHERE persid=manager, dept=deptid, persname="Smith", projid="001"

Query structure before optimisation:

```
PROJECT [projname] (SELECT [projid="001"] (SELECT [persname="Smith"]
(SELECT [dept=deptid] (SELECT [persid=manager] (((Person) TIMES
(Department)) TIMES (Project))))))
```

```
Person
  in:  Person:400:persid,400:persname,350:age,47
  out: 400:persid,400:persname,350:age,47
Department
  in:  Department:5:deptid,5:deptname,5:manager,5
  out: 5:deptid,5:deptname,5:manager,5
(Person) TIMES (Department)
  inl: 400:persid,400:persname,350:age,47
  inr: 5:deptid,5:deptname,5:manager,5
  out: 2000:persid,400:persname,350:age,47:deptid,5:deptname,5:manager,5
Project
  in:  Project:40:projid,40:projname,35:dept,5
  out: 40:projid,40:projname,35:dept,5
((Person) TIMES (Department)) TIMES (Project)
  inl: 2000:persid,400:persname,350:age,47:deptid,5:deptname,5:manager,5
  inr: 40:projid,40:projname,35:dept,5
  out: 80000:persid,400:persname,350:age,47:deptid,5:deptname,5:manager,5:projid,40:projname,35:dept,5
SELECT [persid=manager] (((Person) TIMES (Department)) TIMES (Project))
  in:  80000:persid,400:persname,350:age,47:deptid,5:deptname,5:manager,5:projid,40:projname,35:dept,5
  out: 200:persid,5:persname,200:age,47:deptid,5:deptname,5:manager,5:projid,40:projname,35:dept,5
SELECT [dept=deptid] (SELECT [persid=manager] (((Person) TIMES (Department)) TIMES (Project)))
  in:  200:persid,5:persname,200:age,47:deptid,5:deptname,5:manager,5:projid,40:projname,35:dept,5
  out: 40:persid,5:persname,40:age,40:deptid,5:deptname,5:manager,5:projid,40:projname,35:dept,5
SELECT [persname="Smith"] (SELECT [dept=deptid] (SELECT [persid=manager] (((Person) TIMES (Department))
  in:  40:persid,5:persname,40:age,40:deptid,5:deptname,5:manager,5:projid,40:projname,35:dept,5
  out: 1:persid,1:persname,1:age,1:deptid,1:deptname,1:manager,1:projid,1:projname,1:dept,1
SELECT [projid="001"] (SELECT [persname="Smith"] (SELECT [dept=deptid] (SELECT [persid=manager] (((Perso
  in:  1:persid,1:persname,1:age,1:deptid,1:deptname,1:manager,1:projid,1:projname,1:dept,1
  out: 1:persid,1:persname,1:age,1:deptid,1:deptname,1:manager,1:projid,1:projname,1:dept,1
PROJECT [projname] (SELECT [projid="001"] (SELECT [persname="Smith"] (SELECT [dept=deptid] (SELECT [pers
  in:  1:persid,1:persname,1:age,1:deptid,1:deptname,1:manager,1:projid,1:projname,1:dept,1
  out: 1:projname,1
```

Query structure after optimisation:

```
PROJECT [projname] ((PROJECT [deptid] ((PROJECT [persid] (SELECT
[persname="Smith"] (Person))) JOIN [persid=manager] (PROJECT
[manager,deptid] (Department)))) JOIN [dept=deptid] (PROJECT
[projname,dept] (SELECT [projid="001"] (Project))))
```

```
Person
  in:  Person:400:persid,400:persname,350:age,47
  out: 400:persid,400:persname,350:age,47
SELECT [persname="Smith"] (Person)
  in:  400:persid,400:persname,350:age,47
  out: 1:persid,1:persname,1:age,1
PROJECT [persid] (SELECT [persname="Smith"] (Person))
  in:  1:persid,1:persname,1:age,1
  out: 1:persid,1
Department
  in:  Department:5:deptid,5:deptname,5:manager,5
  out: 5:deptid,5:deptname,5:manager,5
PROJECT [manager,deptid] (Department)
  in:  5:deptid,5:deptname,5:manager,5
  out: 5:manager,5:deptid,5
(PROJECT [persid] (SELECT [persname="Smith"] (Person))) JOIN [persid=manager] (PROJECT [manage
  inl: 1:persid,1
  inr: 5:manager,5:deptid,5
  out: 1:persid,1:manager,1:deptid,1
PROJECT [deptid] ((PROJECT [persid] (SELECT [persname="Smith"] (Person))) JOIN [persid=manage
  in:  1:persid,1:manager,1:deptid,1
  out: 1:deptid,1
Project
  in:  Project:40:projid,40:projname,35:dept,5
  out: 40:projid,40:projname,35:dept,5
SELECT [projid="001"] (Project)
  in:  40:projid,40:projname,35:dept,5
  out: 1:projid,1:projname,1:dept,1
PROJECT [projname,dept] (SELECT [projid="001"] (Project))
  in:  1:projid,1:projname,1:dept,1
  out: 1:projname,1:dept,1
(PROJECT [deptid] ((PROJECT [persid] (SELECT [persname="Smith"] (Person))) JOIN [persid=manage
  inl: 1:deptid,1
  inr: 1:projname,1:dept,1
  out: 1:deptid,1:projname,1:dept,1
PROJECT [projname] ((PROJECT [deptid] ((PROJECT [persid] (SELECT [persname="Smith"] (Person))
  in:  1:deptid,1:projname,1:dept,1
  out: 1:projname,1
```