Query Optimization

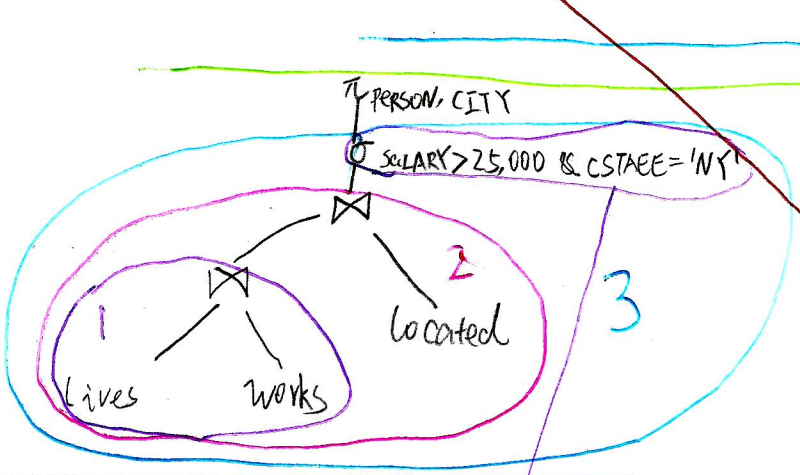
Query Tree

Consider the following schema and query:

Lives(PERSON, STREET, CITY, STATE, ZIP)
works(PERSON, COMPANY, SALARY, POSITION)
located(COMPANY, CCITY, CSTATE, CZIP)

Find the name and city of all people who earn more than 25,000 and work for a company located in the state of NY.

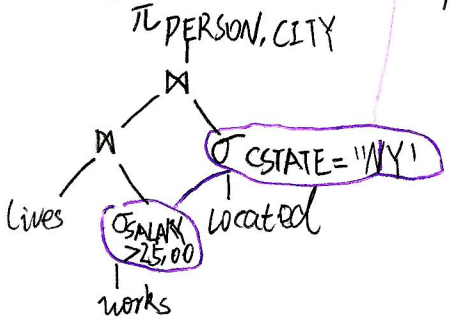
$\pi_{PERSON, CITY}\ \sigma_{SALARY>25,000 \wedge CSTATE='NY'}\ ((lives \bowtie works) \bowtie located)$



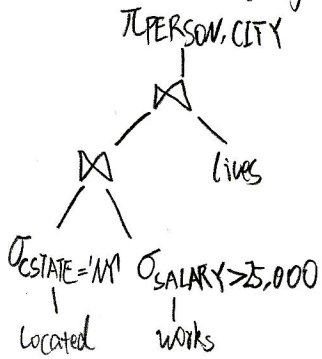
HEURISTIC ALGEBRAIC OPTIMIZATION ALGORITHM

1. Cascade selection (rule 1)
2. Move selection as far down the tree as possible (rules 2, 4, 6, 10)
3. Rearrange leaf node to get smaller intermediate relations (rule 9)
4. Combine a X with σ to yield ⋈, if possible
5. Cascade projections and push down the tree (rules 3, 4, 7, 11)
6. Identify common subexpressions.

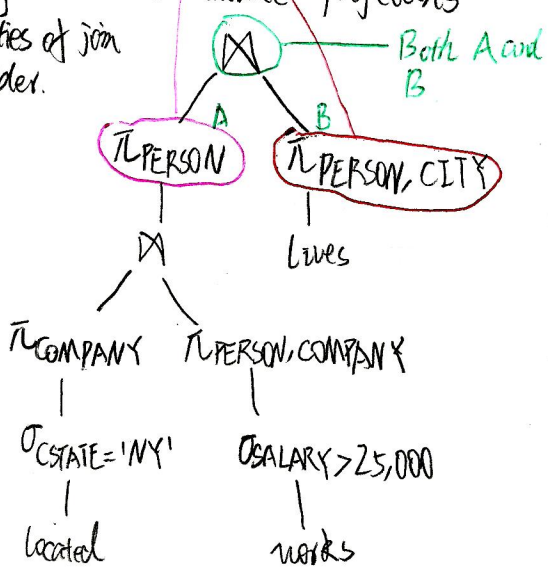
T_1: Cascade selections and push down tree



T_2: Using commutativity & associativity properties of join rearrange join order.



T_3: Introduce projections



Query Optimization

Consider the following relational schema and relational algebra query.

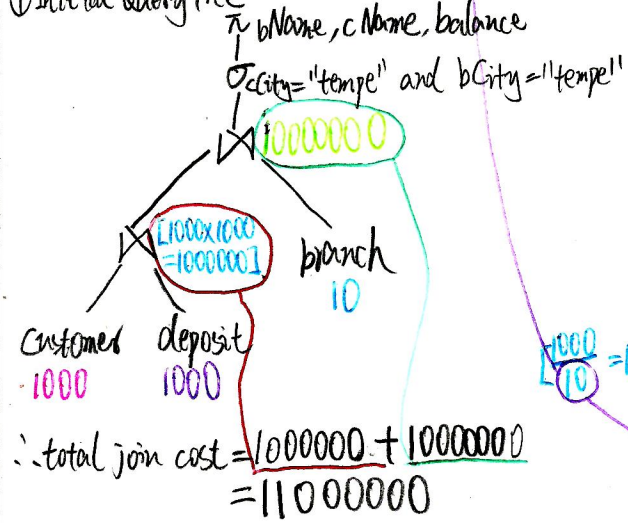
customer (cName, street, cCity)
deposit (cName, bName, accountNumber, balance)
branch (bName, assets, bCity)

Identify a relational algebra tree that reflects the order of operations a decent query optimizer would choose. Use the following statistics to estimate the sizes of intermediate tables and total join costs.

Assume, |customer| = 1000,
|deposit| = 1000,
|branch| = 10,
V(cCity, customer) = 10,
V(bCity, branch) = 10,
Max(balance) = 100K

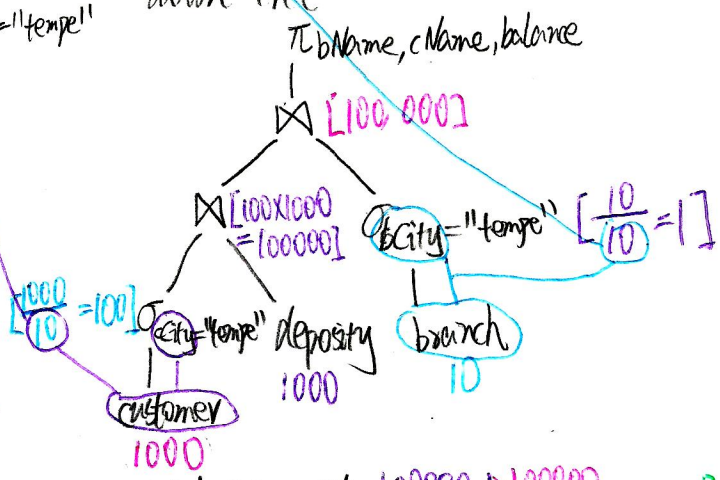
Query:
$\pi_{bName, cName, balance}(\sigma_{cCity = "tempe" \text{ and } bCity = "tempe"}((customer \bowtie deposit) \bowtie branch))$

① Initial Query Tree



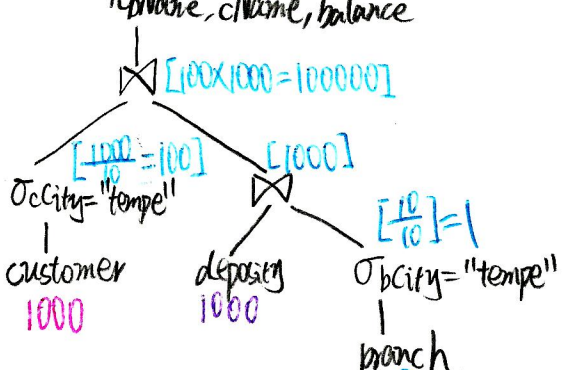
∴ total join cost = 1000000 + 10000000
= 11000000

② Cascade selections and push down tree



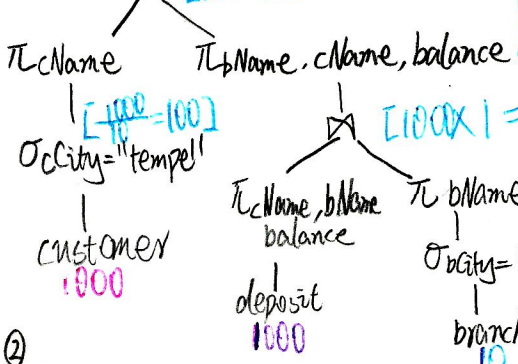
∴ total join cost = 100000 + 100000
= 200000

③ Rearrange join order



∴ total join cost = 100,000 + 1000 = 101,000 < 200,000 ②

④ Break down π



Result

∴ total join cost = 100000 + 1000
= 101000

∴ same as ③ ⇒ We could choose either ③ or ④.