OPTIMIZATION OF RELATIONAL QUERIES

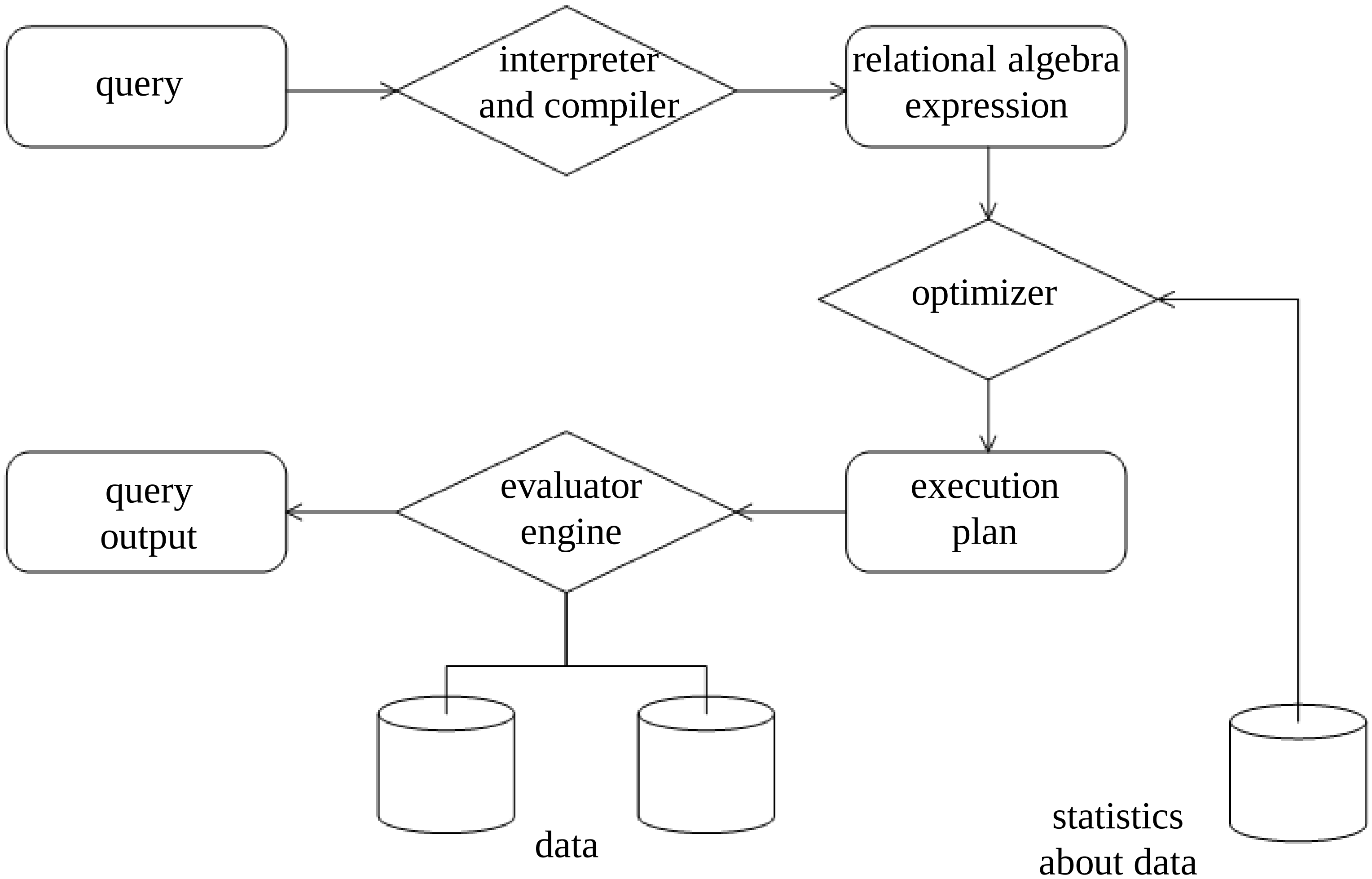
Dr. Gajdos Sándor – Dr. Erős Levente

November 2014 – November 2021

BME–TMIT

CONTENTS

OVERVIEW



query
interpreter
and compiler
relational algebra
expression
optimizer
execution
plan
evaluator
engine
query
output
data
statistics
about data

STEPS

1. (Syntactical) analysis, compilation

   Does it make sense?

   Input: SQL statement

   Result: relational algebra expression

2. Cost optimization

3. Evaluation

CATALOG BASED COST ESTIMATION

- The following statistical data is stored in the so-called catalog
  - Catalog data about relations
  - Catalog data about indexes
  - Query cost
- Cost is estimated based on catalog data.

CATALOG DATA ABOUT RELATIONS

- n_r: number of records in relation r
- b_r: number of blocks storing the records of relation r
- s_r: size of a record
- f_r: how many records fit in a data block

CATALOG DATA ABOUT RELATIONS

- $V(A, r)$: how many different **v**alues attribute A has in relation r (cardinality).
  - $V(A, r) = |\pi_A(r)|$
  - If A is a key, then $V(A, r) = n_r$
- $SC(A, r)$: (**S**election **C**ardinality) average number of records that satisfy a selection condition.
  - If A is a key, then $SC(A, r) = 1$
  - In general $SC(A, r) = \frac{n_r}{V(A,r)}$
- If the records of a relation are physically stored together, then:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

CATALOG DATA ABOUT INDEXES

- f_i: number of pointers going out of a node in case of a tree index, like B* tree
- HT_i: number of index levels (**H**eight of **T**ree)
  - $HT_i = \lceil \log_{f_i} b_r \rceil$ (B* tree)
  - $HT_i = 1$ (hash)
- LB_i: a number of leaf blocks (**L**owest level index **B**lock)

COST OF QUERY

Definition:

- **Number of block reading and writing operations from the disc** (without writing out the result).
- Takes the most time by far – good metric
  - By orders of magnitude more costly then performing operations in the RAM, etc.

COST OF OPERATIONS – OUTLINE

- Selection
  - Selection algorithms (basic, indexed, comparison based)
  - complex selection
- Join
  - Types
  - Size estimation
  - Join algorithms
- Other
  - Filtering repetitions
  - Union, intersection, subtraction

BASIC SELECTION ALGORITHMS (=)

A1: Linear search

- Cost:

$$E_{A1} = b_r$$

A2: Binary search

- Requirements:
  - Blocks are located continuously on the disk
  - The file is ordered by attribute A
  - The selection condition is equality on attribute A
- Cost:

$$E_{A2} = \lceil \log_2(b_r + 1) \rceil + \left\lceil \frac{SC(A,r)}{f_r} \right\rceil - 1$$

INDEXED SEARCH ALGORITHMS

Primary index – requires the data file to be physically ordered by the index attribute (search key). Everything else is a **secondary index**

A3: Using primary index, if the equality condition is defined on the key

$$\text{Cost: } E_{A3} = HT_i + 1$$

A4: Using primary index, if the equality condition is defined on a non-key attribute (the primary index is on the non-key attribute)

$$\text{Cost: } E_{A4} = HT_i + \left\lceil \frac{SC(A,r)}{f_r} \right\rceil$$

A5: Using secondary index.

$$\text{Cost: } E_{A5} = HT_i + SC(A,r)$$

$$\text{Cost: } E_{A5} = HT_i + 1, \text{ if } A \text{ is a key}$$

COMPARISON BASED SELECTION – $\sigma_{A \leq v}(R)$

Estimation of the **number of result records**:

- If v is unknown: $\frac{n_r}{2}$

- If v is known, and the distribution is uniform:

$$n_{\text{average}} = n_r \cdot \frac{v - \min(A, r)}{\max(A, r) - \min(A, r)}$$

COMPARISON BASED SELECTION – $\sigma_{A \leq v}(R)$

A6: With primary index.

- If v is unknown:

$$\text{Cost: } E_{A6} = HT_i + \frac{b_r}{2}$$

- If v is known:

$$\text{Cost: } E_{A6} = HT_i + \left\lceil \frac{c}{f_r} \right\rceil,$$

Where c is the number of records for which $A \leq v$

A7: With secondary index

$$\text{Cost: } E_{A7} = HT_i + \frac{LB_i}{2} + \frac{n_r}{2}$$

JOIN OPERATION

Definition:

$$r_1 \bowtie_\theta r_2 = \sigma_\theta(r_1 \times r_2)$$

Types:

- Natural join

$$r_1 \bowtie r_2 = \pi_{A \cup B}\left(\sigma_{R1.X=R2.X}\,(r_1 \times r_2)\right)$$

- Outer join
  - Left outer join: $r_1 * (+) r_2$
  - Right outer join: $r_1 (+) * r_2$
  - Full outer join: $r_1 (+) * (+) r_2$

- Theta join:

$$r_1 \bowtie_\theta r_2 = \sigma_\theta(r_1 \times r_2)$$

JOIN OPERATION: NESTED LOOP JOIN

Given are two relations, r and s:

FOR each record $t_r \in r$ DO BEGIN

  FOR each record $t_s \in s$ DO BEGIN

    test if pair (t_r, t_s), fulfills join condition θ

    IF yes, THEN add record $t_r . t_s$ to the result

  END

END

- „worst case” cost: $n_r \cdot b_s + b_r$
- If at least one relation fits in the memory, then its cost is: $b_r + b_s$

JOIN OPERATION: BLOCK NESTED LOOP JOIN

```
FOR each block b_r ∈ r DO BEGIN
    FOR each block b_s ∈ s DO BEGIN
        FOR each record t_r ∈ b_r DO BEGIN
            FOR each record t_s ∈ b_s DO BEGIN
                test pair (t_r, t_s)
            END
        END
    END
END
```

- „worst-case” cost: $b_r \cdot b_s + b_r$
- with a lot of memory: $b_r + b_s$

JOIN OPERATION: INDEXED NESTED LOOP JOIN

For one of the relations (s) we have an index

Let us put the indexed relation to the inner cycle of the first algorithm

$\Longrightarrow$ Using the index, the search can be performed at a lower cost

Cost:

$$b_r + n_r \cdot c,$$

where c is the cost of selection on s.

FURTHER JOIN IMPLEMENTATIONS

- sorted merge join
  - order relations by the attributes provided in the join condition
- hash join
  - one of the relations is accessed through a hash table when looking for its records matching the records of the other relation

FURTHER OPERATIONS

- *Filtering repetitions* (ordering, and then deleting)
- *Projection* (projection, then filtering repetitions)
- *Union* (ordering of both relations, and filtering duplications during merge)
- *Intersection* (ordering both relations, then leaving only the repetitions during merge)
- *Subtraction* (ordering both relations, and then during merge, we only leave those elements in the result set, which are only present in the first relation)

METHODS FOR EVALUATING EXPRESSIONS

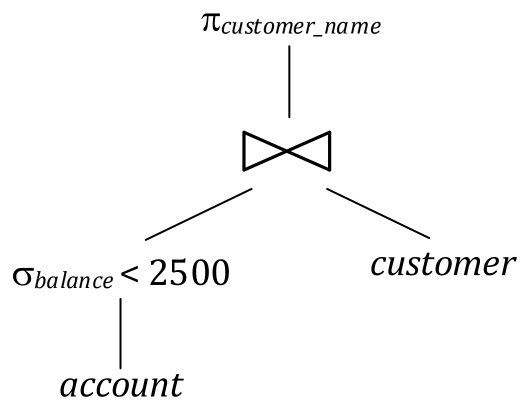
- Materialization
  - Evaluating a single operation of a complex expression at a time
- Pipelining
  - Multiple operations are evaluated in parallel
  - The result of an opeartion is immediately transferred to the input of the next operation

METHODS FOR EVALUATING EXPRESSIONS: MATERIALIZATION

- Canonical format:

$$\pi_{customer\_name}(\sigma_{balance<2500}(account) \bowtie customer)$$

- Query tree:



- Final cost: cost of operations + cost of storing sub-results
- Advantage: Easy implementation
- Drawback: Many storage operations (block operations)

METHODS FOR EVALUATING EXPRESSIONS: PIPELINING

- Parallel evaluation
- Operations create sub-results for the following operation, based on the sub-results of the preceding operation
- Does not calculate the whole relation (sub-result) at a time

Advantage:

- No temporary storage needed
- Low memory requirement

Drawback:

- Narrows down the algorithms that can be used

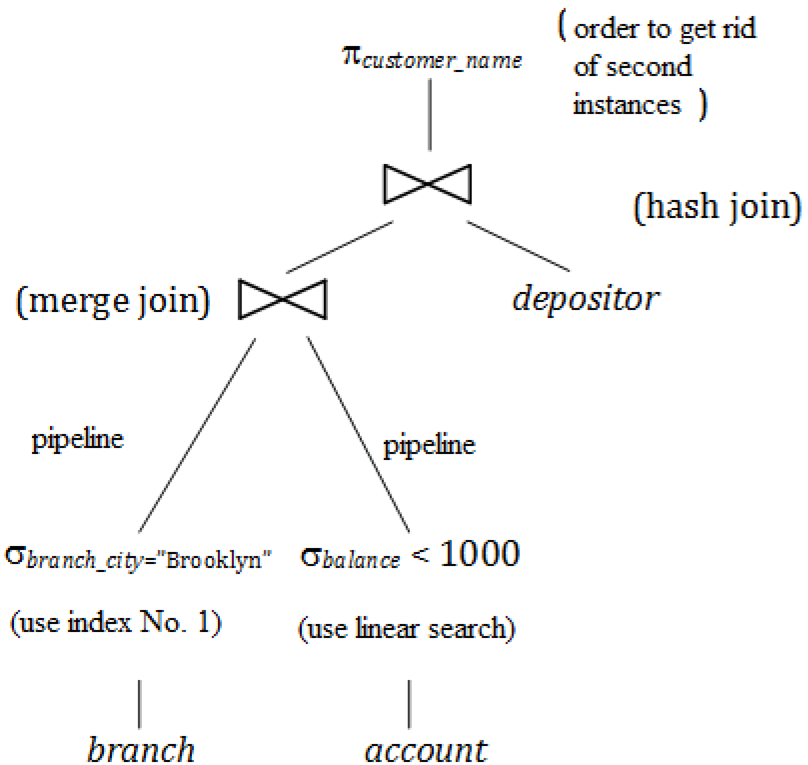
CHOOSING THE EXECUTION PLAN

Many execution plans are possible for the same result

Questions to be answered:

- Which operations?
- In what order?
- By which algorithm?
- By which workflow?

One exact execution plan:



CHOOSING THE EXECUTION PLAN: COST-BASED OPTIMIZATION

Greedy and wrong strategy:

- Listing all possible equivalent execution plans
- Evaluating each plan
- Choosing the optimal one

Example: In case of expression $r_1 \bowtie r_2 \bowtie r_3 \rightarrow 12$ equivalent expressions

In more general: to join n relations, $\frac{(2(n-1))!}{(n-1)!}$ equivalent expressions exist.

This would mean too much load for the system.

Solution: Heuristic cost-based optimization

CHOOSING THE EXECUTION PLAN: HEURISTIC, RULE BASED OPTIMIZATION

- Manipulating the query tree
- Example:

EMPLOYEE (EMPLOYEE_ID, LAST_NAME, FIRST_NAME, BIRTH_DATE, …)

PROJECT (PROJECT_ID, PNAME, …)

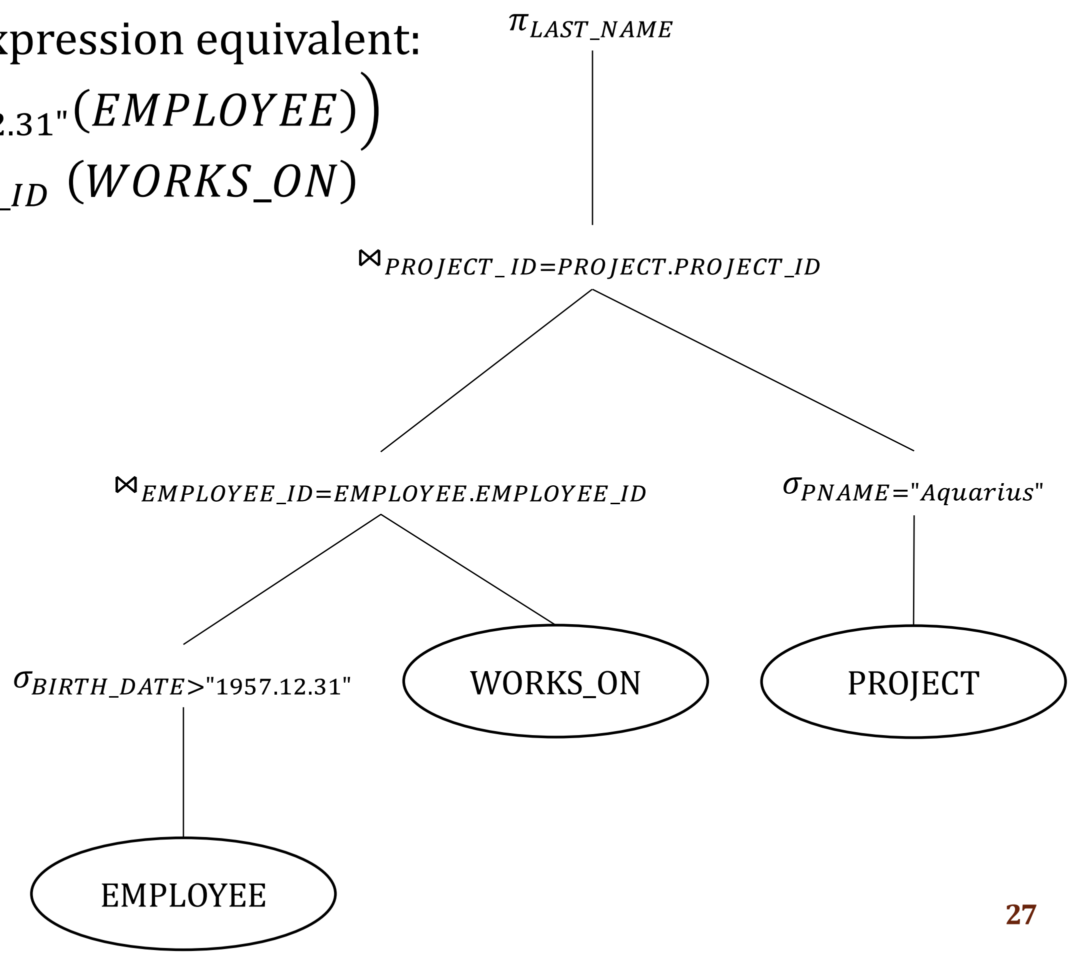
WORKS_ON (PROJECT_ID, EMPLOYEE_ID)

```
select last_name
  from employee, works_on, project
 where employee.birth_date > '1957.12.31'
   and works_on.project_id = project.project_id
   and works_on.employee_id = employee.employee_id
   and project.pname = 'Aquarius'
```

CHOOSING THE EXECUTION PLAN: HEURISTIC, RULE BASED OPTIMIZATION

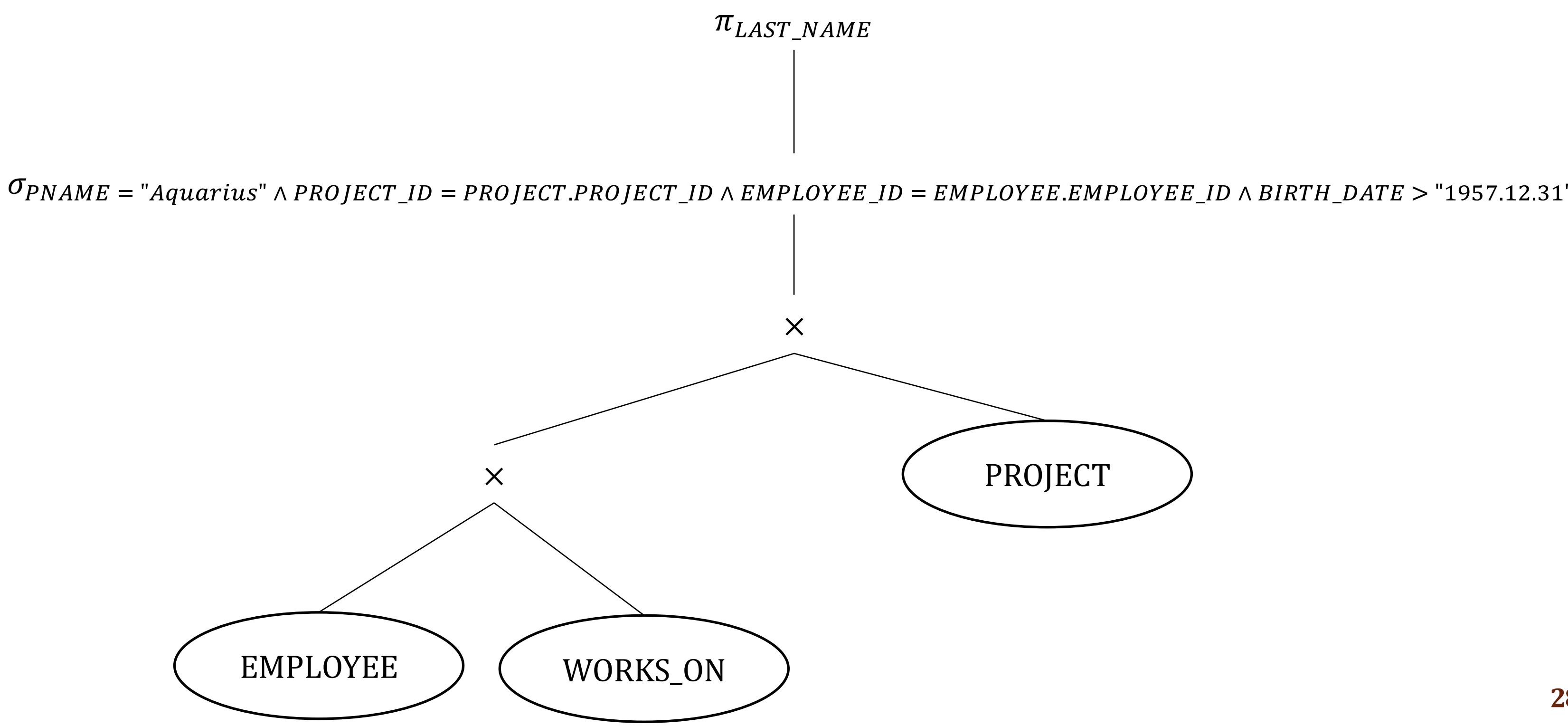
- A possible relational algebra expression equivalent:

$$\pi_{LAST\_NAME}\left(\left(\sigma_{BIRTH\_DATE>"1957.12.31"}(EMPLOYEE)\right) \bowtie_{EMPLOYEE\_ID=EMPLOYEE.EMPLOYEE\_ID} (WORKS\_ON) \bowtie_{PROJECT\_ID=PROJECT.PROJECT\_ID} \sigma_{PNAME="Aquarius"}(PROJECT)\right)$$



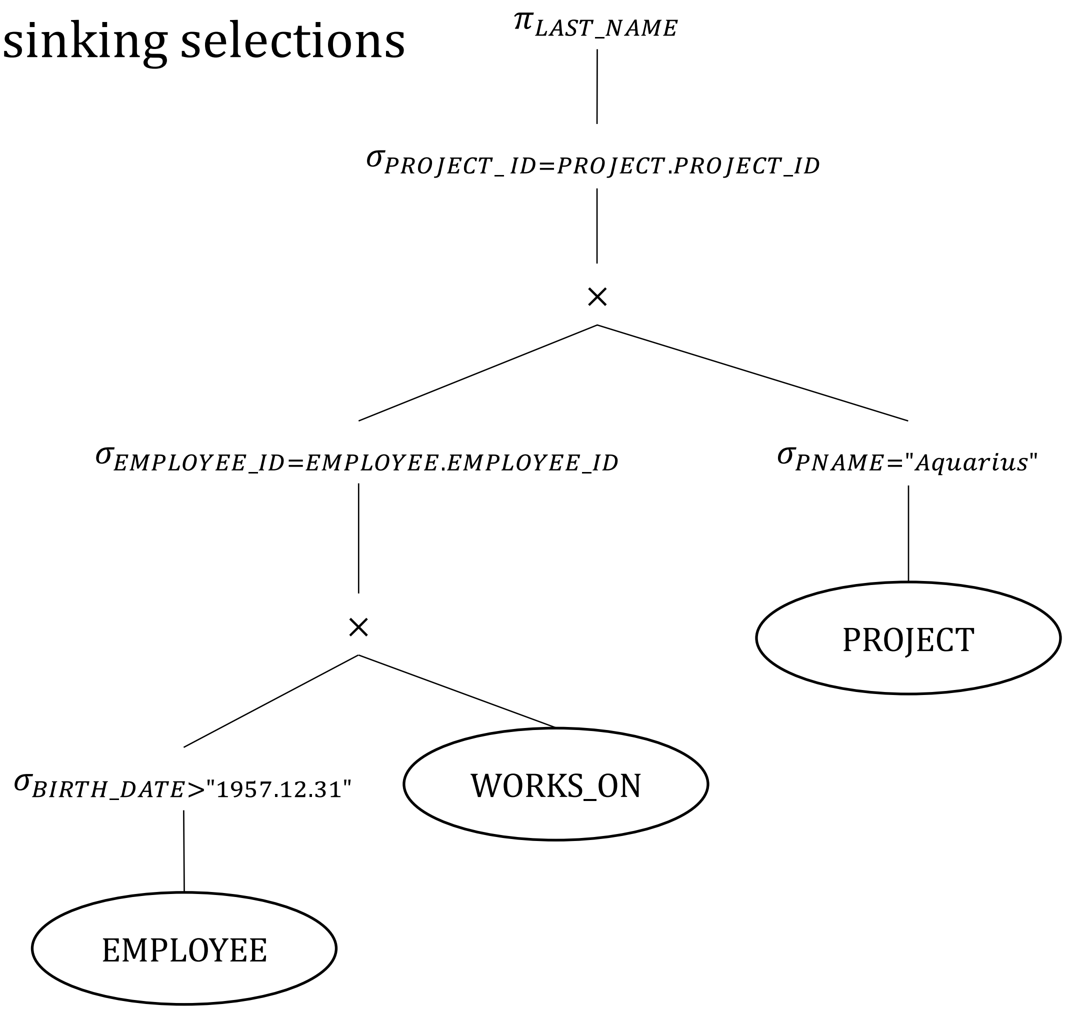
CHOOSING THE EXECUTION PLAN: HEURISTIC, RULE BASED OPTIMIZATION

- Goal: choosing the quickest equivalent
- Step 1: canonical format (Cartesian, selection, projection)



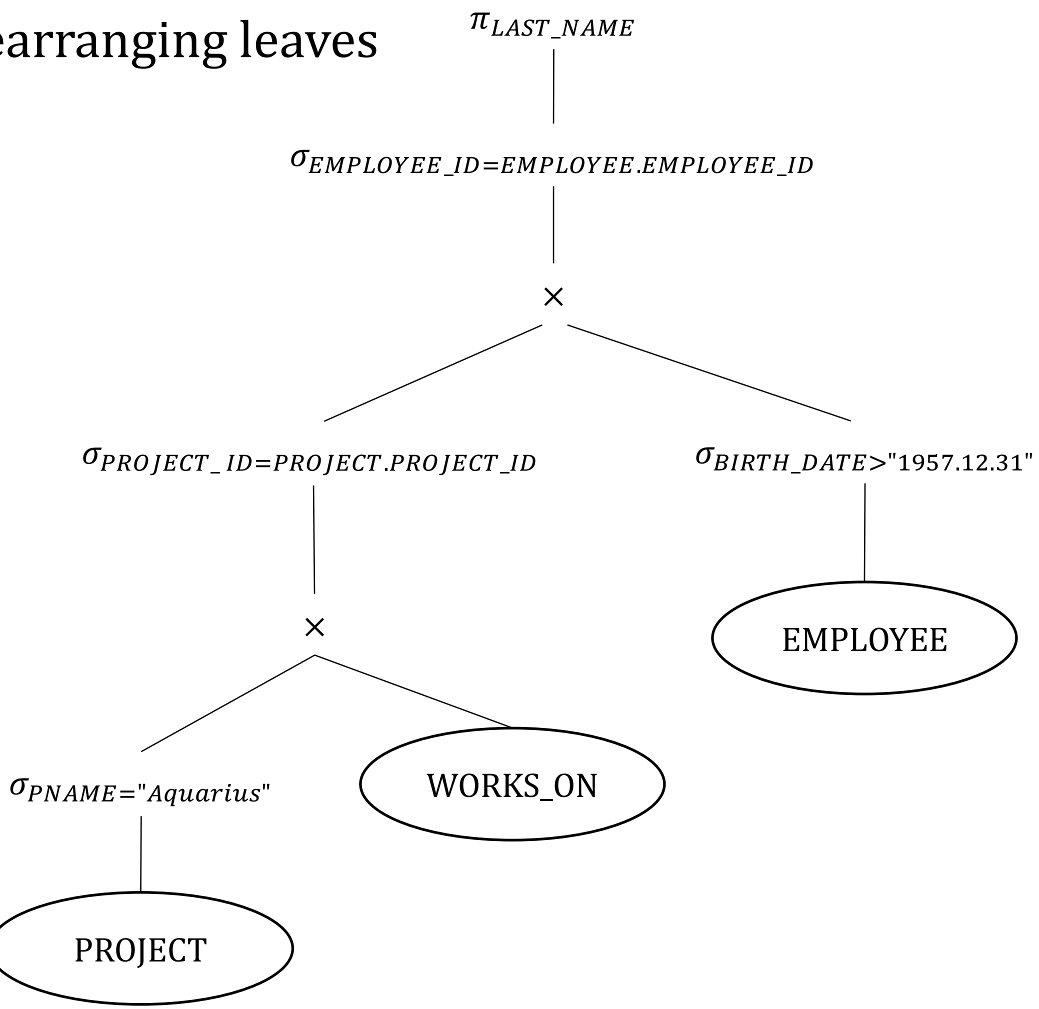
CHOOSING THE EXECUTION PLAN: HEURISTIC, RULE BASED OPTIMIZATION

- Step 2: sinking selections



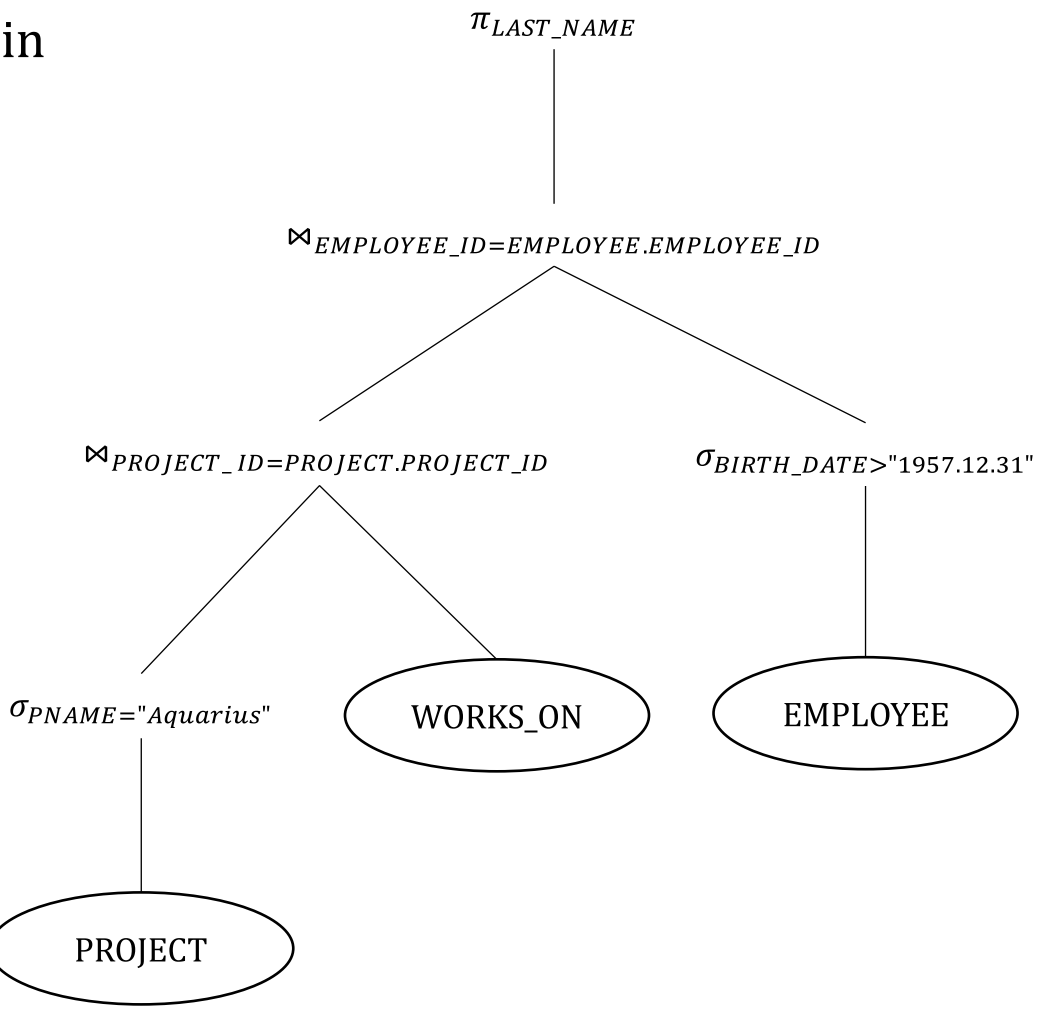
CHOOSING THE EXECUTION PLAN: HEURISTIC, RULE BASED OPTIMIZATION

- Step 3: rearranging leaves



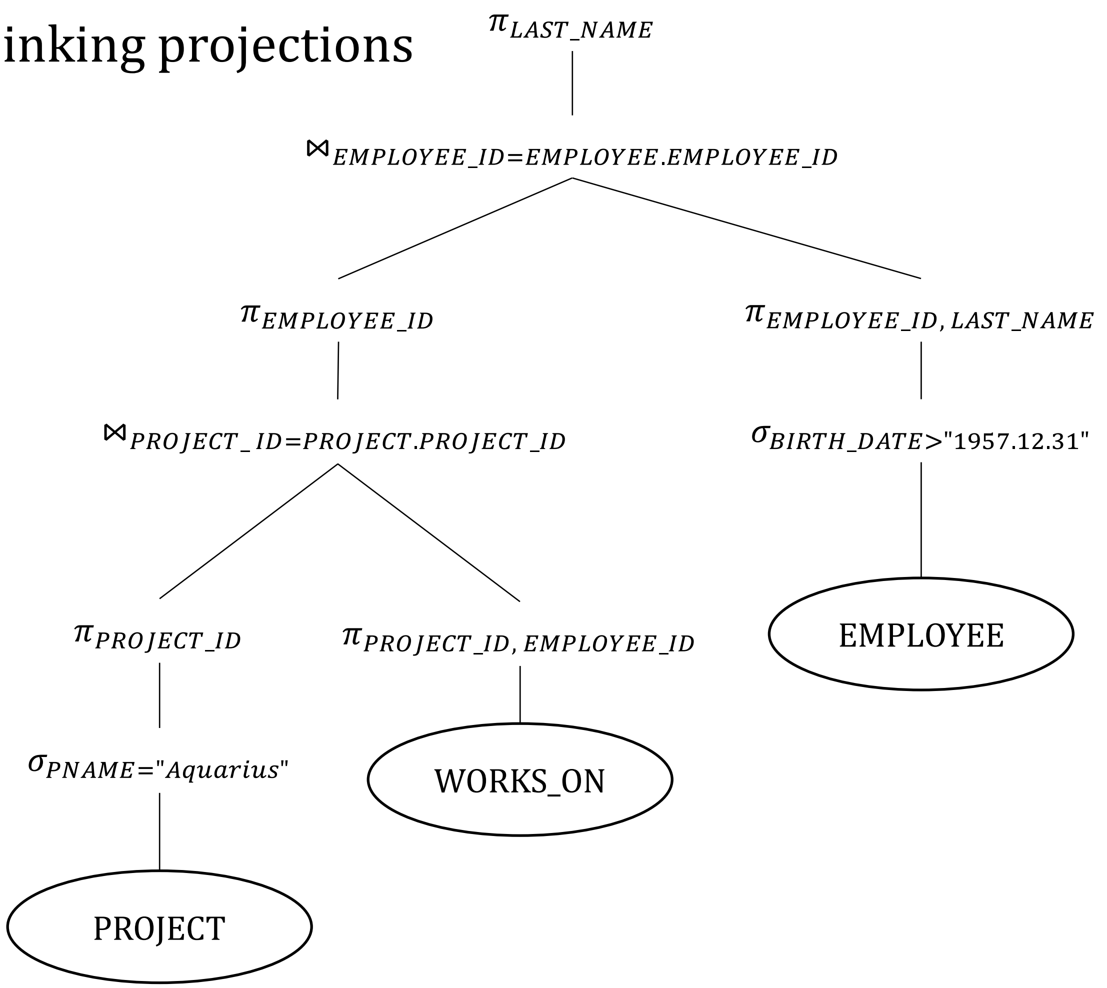
CHOOSING THE EXECUTION PLAN: HEURISTIC, RULE BASED OPTIMIZATION

- Step 4: join



CHOOSING THE EXECUTION PLAN: HEURISTIC, RULE BASED OPTIMIZATION

- Step 5: sinking projections



WHEN ARE TWO TREES EQUIVALENT?

RELATIONAL ALGEBRA TRANSFORMATIONS I.

- $\sigma_{c_1 \wedge c_2 \wedge \cdots \wedge c_n}(r) \equiv \sigma_{c_1}\left(\sigma_{c_2}\left(\ldots\left(\sigma_{c_n}(r)\right)\ldots\right)\right)$
- $\sigma_{c_1}\left(\sigma_{c_2}(r)\right) \equiv \sigma_{c_2}\left(\sigma_{c_1}(r)\right)$
- $\pi_{List_1}\left(\pi_{List_2}\left(\ldots\left(\pi_{List_n}(r)\right)\ldots\right)\right) \equiv \pi_{List_1}(r)$
- $\pi_{A_1, A_2, \ldots, A_n}(\sigma_c(r)) \equiv \sigma_c\left(\pi_{A_1, A_2, \ldots, A_n}(r)\right)$

WHEN ARE TWO TREES EQUIVALENT?

RELATIONAL ALGEBRA TRANSFORMATIONS II.

- $r \bowtie_c s \equiv s \bowtie_c r$
- $\sigma_c(r \bowtie s) \equiv \left(\sigma_c(r)\right) \bowtie s$
- $\pi_L(r \bowtie_c s) \equiv \left(\pi_{A_1,\dots,A_n}(r)\right) \bowtie_c \left(\pi_{B_1,\dots,B_m}(s)\right)$
- $\pi_L(r \bowtie_c s) \equiv$ $\pi_L\left(\left(\pi_{A_1,\dots,A_n,A_{n+1},\dots,A_{n+k}}(r)\right) \bowtie_c \left(\pi_{B_1,\dots,B_m,B_{m+1},\dots,B_{m+p}}(s)\right)\right)$

Set operations (union, intersection) are commutative

Join, Cartesian product, union, and intersection are associative:

$$(r\theta s)\theta t \equiv r\theta(s\theta t)$$

WHEN ARE TWO TREES EQUIVALENT?

RELATIONAL ALGEBRA TRANSFORMATIONS III.

- $\sigma_C(r\ \theta\ s) \equiv \left(\sigma_C(r)\right)\ \theta\ \left(\sigma_C(s)\right)$
- $\pi_L(r\ \theta\ s) \equiv \left(\pi_L(r)\right)\ \theta\ (\pi_L(s))$

Further rules:

- $c \equiv \neg(c_1 \wedge c_2) \equiv (\neg c_1\ ) \vee (\neg c_2)$
- $c \equiv \neg(c_1 \vee c_2) \equiv (\neg c_1\ ) \wedge (\neg c_2)$

RULES, SUMMARY

- Conjuctions in selections are transformed to a series of selections.
- Selections are swapped with the other operations.
- Query tree leaves are re-arranged.
- Cartesian products and the selection (join) condition above them are transformed to a single operation (theta join)
- Projections are swapped with the other operations