

Optimization of Logical Queries

Task:

Consider the following relational schema:

- Emp(eid, did, sal, hobby)
- Dept(did, dname, floor, phone)
- Finance(did, budget, sales, expenses)

For the following SQL statement:

1. Translate the query into the relational algebra.
2. Remove redundant joins from the select-project-join subexpressions in the obtained logical query plan.
3. By means of the algebraic laws, further optimize the obtained expression.

Optimization of Logical Queries

Task (continued)

```
SELECT D.floor
FROM Dept D, Emp E
WHERE
  (D.floor = 1
   OR D.floor IN
     ( SELECT D2.floor FROM Dept D2, Finance F1
       WHERE F1.budget > 150 AND D2.did = F1.did)
  )
AND E.did = D.did
AND E.did IN (SELECT F2.did FROM Finance F2, Emp E2
              WHERE F2.did = E.did AND E2.did = D.did
              AND E2.eid = E.eid AND F2.expenses = 300)
```

Optimization of Logical Queries

Solution: translation into the relational algebra

First, we normalize the query to a form with only EXISTS and NOT EXISTS subqueries:

```
SELECT D.floor
FROM Dept D, Emp E
WHERE
  (D.floor = 1 OR EXIST
    ( SELECT D2.floor FROM Dept D2, Finance F1
      WHERE F1.budget > 150 AND D2.did = F1.did
        AND D2.floor = D.floor) )
AND E.did = D.did
AND EXISTS (SELECT F2.did FROM Finance F2, Emp E2
  WHERE F2.did = E.did AND E2.did = D.did
    AND E2.eid = E.eid AND F2.expenses = 300
    AND E.did = F2.did)
```

Optimization of Logical Queries

Conjunctive Normal Form

```
SELECT D.floor
FROM Dept D, Emp E
WHERE ( D.floor = 1
      AND E.did = D.did
      AND EXISTS (SELECT F2.did FROM Finance F2, Emp E2
                  WHERE F2.did = E.did AND E2.did = D.did
                  AND E2.eid = E.eid AND F2.expenses = 300 AND E.did = F2.did)
) OR (
  EXIST ( SELECT D2.floor FROM Dept D2, Finance F1
          WHERE F1.budget > 150 AND D2.did = F1.did
          AND D2.floor = D.floor)
  AND E.did = D.did
  AND EXISTS (SELECT F2.did FROM Finance F2, Emp E2
              WHERE F2.did = E.did AND E2.did = D.did
              AND E2.eid = E.eid AND F2.expenses = 300 AND E.did = F2.did) )
```

Optimization of Logical Queries

Normalize to UNION

```
Q1 = SELECT D.floor
      FROM Dept D, Emp E
      WHERE D.floor = 1
            AND E.did = D.did
            AND EXISTS (SELECT F2.did FROM Finance F2, Emp E2
                        WHERE F2.did = E.did AND E2.did = D.did
                        AND E2.eid = E.eid AND F2.expenses = 300
                        AND E.did = F2.did)
```

Optimization of Logical Queries

Normalize to UNION

```
Q2 = SELECT D.floor
FROM Dept D, Emp E
WHERE
  EXIST ( SELECT D2.floor FROM Dept D2, Finance F1
          WHERE F1.budget > 150 AND D2.did = F1.did
          AND D2.floor = D.floor)
AND E.did = D.did
AND EXISTS (SELECT F2.did FROM Finance F2, Emp E2
            WHERE F2.did = E.did AND E2.did = D.did
            AND E2.eid = E.eid AND F2.expenses = 300
            AND E.did = F2.did)
```

The new query is Q1 UNION Q2.

Optimization of Logical Queries

Translation of the innermost subqueries

```
SELECT F2.did FROM Finance F2, Emp E2
      WHERE F2.did = E2.did AND E2.did = D.did
      AND E2.eid = E.eid AND F2.expenses = 300
      AND E.did = F2.did
```

This subquery is translated as follows:

$$e_1 = \pi_{F_2.did, E.*, D.*} \sigma_{F_2.did=E2.did \wedge E2.did=D.did \wedge E2.eid=E.eid} \\ \sigma_{F_2.expenses=300 \wedge E.did=F_2.did} (\rho_D(\text{Dept}) \times \rho_E(\text{Emp}) \times \rho_{F_2}(\text{Finance}) \times \rho_{E_2}(\text{Emp}))$$

Optimization of Logical Queries

Translation of the innermost subqueries

```
SELECT D2.floor FROM Dept D2, Finance F1
WHERE F1.budget > 150 AND D2.did = F1.did
AND D2.floor = D.floor
```

This subquery is translated as follows:

$$e_2 = \pi_{D_2.floor, D.*} \sigma_{F_1.budget > 150 \wedge D_2.did = F_1.did} \\ \sigma_{D_2.floor = D.floor} (\rho_D(\text{Dept}) \times \rho_{D_2}(\text{Dept}) \times \rho_{F_1}(\text{Finance}))$$

Optimization of Logical Queries

Translation of the Middle Queries

```
Q1 = SELECT D.floor FROM Dept D, Emp E
      WHERE D.floor = 1 AND E.did = D.did
      AND EXISTS (SELECT F2.did FROM Finance F2, Emp E2
                  WHERE F2.did = E.did AND E2.did = D.did
                  AND E2.eid = E.eid AND F2.expenses = 300
                  AND E.did = F2.did)
```

The translation of the from part gives

$$e_3 = (\rho_D(\text{Dept}) \times \rho_E(\text{Emp}))$$

To de-correlate we compute:

$$f = \hat{e}_3 \bowtie \pi_{D.*,E.*}(e_1)$$

Note that \hat{e}_3 is empty and hence

$$f = \pi_{D.*,E.*}(e_1)$$

To this expression we add the WHERE and SELECT clause:

$$e_4 = \pi_{D.floor}(\sigma_{D.floor=1 \wedge E.did=D.did}(\pi_{D.*,E.*}(e_1)))$$

Optimization of Logical Queries

Translation of the Middle Queries

```
Q2 = SELECT D.floor
      FROM Dept D, Emp E
      WHERE
        EXIST ( SELECT D2.floor FROM Dept D2, Finance F1
                WHERE F1.budget > 150 AND D2.did = F1.did
                AND D2.floor = D.floor)
        AND E.did = D.did
        AND EXISTS (SELECT F2.did FROM Finance F2, Emp E2
                    WHERE F2.did = E.did AND E2.did = D.did
                    AND E2.eid = E.eid AND F2.expenses = 300
                    AND E.did = F2.did)
```

The translation of the from part gives

$$e_5 = (\rho_D(\text{Dept}) \times \rho_E(\text{Emp}))$$

To de-correlate we compute:

$$f' = \hat{e}_5 \bowtie (\pi_{D.*,E.*}(e_1) \bowtie \pi_{D.*}(e_2)) = (\pi_{D.*,E.*}(e_1) \bowtie \pi_{D.*}(e_2))$$

To this expression we add the WHERE and SELECT clause:

$$e_6 = \pi_{D.floor} \sigma_{E.did=D.did} (\pi_{D.*,E.*}(e_1) \bowtie \pi_{D.*}(e_2))$$

Optimization of Logical Queries

Translation of the Whole Query

Q1 UNION Q2

Since the schemas of e_4 and e_6 are the same, the union is straightforward:

$$e = e_4 \cup e_6$$

Written in full:

$$\begin{aligned} e = & \pi_{D.floor} \sigma_{D.floor=1 \wedge E.did=D.did} \\ & \pi_{D.*,E.*} \sigma_{F_2.did=E.did \wedge E_2.did=D.did \wedge E_2.eid=E.eid \wedge F_2.expenses=300 \wedge E.did=F_2.did} \\ & (\rho_D(\text{Dept}) \times \rho_E(\text{Emp}) \times \rho_{F_2}(\text{Finance}) \times \rho_{E_2}(\text{Emp})) \\ \cup \\ & \pi_{D.floor} \sigma_{E.did=D.did} (\\ & [\pi_{D.*,E.*} \sigma_{F_2.did=E.did \wedge E_2.did=D.did \wedge E_2.eid=E.eid \wedge F_2.expenses=300 \wedge E.did=F_2.did} \\ & (\rho_D(\text{Dept}) \times \rho_E(\text{Emp}) \times \rho_{F_2}(\text{Finance}) \times \rho_{E_2}(\text{Emp}))] \\ & \bowtie [\pi_{D.*} \sigma_{F_1.budget>150 \wedge D_2.did=F_1.did \wedge D_2.floor=D.floor} \\ & (\rho_D(\text{Dept}) \times \rho_{D_2}(\text{Dept}) \times \rho_{F_1}(\text{Finance}))]) \end{aligned}$$

Optimization of Logical Queries

Redundant Joins Removal

The query comprises the following maximal select-project-join subexpressions:

- $\pi_{D.floor} \sigma_{D.floor=1 \wedge E.did=D.did} \pi_{D.*,E.*} \sigma_{...} (\rho_D(\text{Dept}) \times \rho_E(\text{Emp}) \times \rho_{F2}(\text{Finance}) \times \rho_{E2}(\text{Emp}))$
- $[\pi_{D.*,E.*} \sigma_{...} (\rho_D(\text{Dept}) \times \rho_E(\text{Emp}) \times \rho_{F2}(\text{Finance}) \times \rho_{E2}(\text{Emp}))]$
- $(\rho_D(\text{Dept}) \times \rho_{D2}(\text{Dept}) \times \rho_{F1}(\text{Finance}))$

Note that “ $F_1.budget > 150$ ” cannot be included in a select-project-join expression. Also note that the third expression does not contain redundant joins (Why?).

Optimization of Logical Queries

Redundant Joins Removal

The first expression corresponds to:

$$Q_1("1") \leftarrow \text{Dept}(a_1, a_2, "1", a_4), \text{Emp}(b_1, a_1, b_3, b_4), \text{Finance}(a_1, c_2, c_3, "300"), \\ \text{Emp}(b_1, a_1, d_3, d_4)$$

The first and third atoms cannot be removed (Why?)

We check whether we can remove the second atom:

$$Q_2("1") \leftarrow \text{Dept}(a_1, a_2, "1", a_4), \text{Finance}(a_1, c_2, c_3, "300"), \text{Emp}(b_1, a_1, d_3, d_4)$$

The corresponding canonical database: $D_2("1") =$
 $\{\text{Dept}(a_1, a_2, "1", a_4), \text{Finance}(a_1, c_2, c_3, "300"), \text{Emp}(b_1, a_1, d_3, d_4)\}$

Clearly $("1") \in Q_1(D_2)$ because of the matching

$$\begin{array}{lll} a_1 \mapsto a_1 & a_2 \mapsto a_2 & a_4 \mapsto a_4 \\ b_1 \mapsto b_1 & b_3 \mapsto b_3 & b_4 \mapsto b_4 \\ c_2 \mapsto c_2 & c_3 \mapsto c_3 & d_3 \mapsto b_3 \quad d_4 \mapsto b_4 \end{array}$$

hence $Q_2 \subseteq Q_1$. The other direction always holds. Hence $Q_1 \equiv Q_2$

Optimization of Logical Queries

Redundant Joins Removal

No other atom can be removed (Why?).

The optimal query is hence

$Q_2(\text{"1"}) \leftarrow \text{Dept}(a_1, a_2, \text{"1"}, a_4), \text{Finance}(a_1, c_2, c_3, \text{"300"}), \text{Emp}(b_1, a_1, d_3, d_4)$

Translating this query back to the relational algebra, we obtain:

$$\pi_{D.\text{floor}} \left(\left[\sigma_{D.\text{floor}=1 \wedge E_2.\text{did}=D.\text{did} \wedge F_2.\text{did}=E_2.\text{did} \wedge E_2.\text{did}=D.\text{did} \wedge F_2.\text{expenses}=300} \right. \right. \\ \left. \left. (\rho_D(\text{Dept}) \times \rho_{F_2}(\text{Finance}) \times \rho_{E_2}(\text{Emp})) \right] \right)$$

Optimization of Logical Queries

Redundant Joins Removal

The second expression is:

$$\left[\pi_{D.*,E.*} \sigma_{F_2.did=E.did \wedge E_2.did=D.did \wedge E_2.eid=E.eid \wedge F_2.expenses=300 \wedge E.did=F_2.did} (\rho_D(\text{Dept}) \times \rho_E(\text{Emp}) \times \rho_{F_2}(\text{Finance}) \times \rho_{E_2}(\text{Emp})) \right]$$

Translated:

$$Q_3(a_1, a_2, a_3, a_4, b_1, b_3, b_4) \leftarrow \text{Dept}(a_1, a_2, a_3, a_4), \text{Emp}(b_1, a_1, b_3, b_4), \\ \text{Finance}(a_1, c_2, c_3, \text{"300"}), \text{Emp}(b_1, a_1, d_3, d_4)$$

We cannot remove the second atom, this time (why?)

Optimization of Logical Queries

Redundant Joins Removal

However, with a similar mapping as for the first expression, the fourth atom can be removed, and we obtain:

$$Q_4(a_1, a_2, a_3, a_4, b_1, b_3, b_4) \leftarrow \text{Dept}(a_1, a_2, a_3, a_4), \text{Emp}(b_1, a_1, b_3, b_4), \\ \text{Finance}(a_1, c_2, c_3, \text{"300"})$$

We have thus $Q_4 \subseteq Q_3$. The other direction always holds. Hence $Q_3 \equiv Q_4$.

Translating this query back to the relational algebra, we obtain:

$$\left[\pi_{D.*,E.*} \sigma_{F_2.\text{did}=E.\text{did} \wedge E.\text{did}=D.\text{did} \wedge F_2.\text{expenses}=300 \wedge E.\text{did}=F_2.\text{did}} \right. \\ \left. (\rho_D(\text{Dept}) \times \rho_E(\text{Emp}) \times \rho_{F_2}(\text{Finance})) \right]$$

Optimization of Logical Queries

Redundant Joins Removal

The third expression is:

$$(\rho_D(\text{Dept}) \times \rho_{D2}(\text{Dept}) \times \rho_{F1}(\text{Finance}))$$

Translated:

$$Q_5(a_1, \dots, a_4, b_1, \dots, b_4, c_1, \dots, c_4) \leftarrow \text{Dept}(a_1, a_2, a_3, a_4), \text{Dept}(b_1, b_2, b_3, b_4), \\ \text{Finance}(c_1, c_2, c_3, c_4)$$

No atoms can be removed (why?)

Optimization of Logical Queries

Redundant Joins Removal

The optimized expression is therefore:

$$\begin{aligned} e = & \pi_{D.floor}([\sigma_{D.floor=1 \wedge E_2.did=D.did \wedge F_2.did=E_2.did \wedge E_2.did=D.did \wedge F_2.expenses=300} \\ & (\rho_D(\text{Dept}) \times \rho_{F_2}(\text{Finance}) \times \rho_{E_2}(\text{Emp}))]) \\ & \cup \\ & \pi_{D.floor}(\\ & [\pi_{D.*,E.*} \sigma_{F_2.did=E.did \wedge E.did=D.did \wedge F_2.expenses=300 \wedge E.did=F_2.did} \\ & (\rho_D(\text{Dept}) \times \rho_E(\text{Emp}) \times \rho_{F_2}(\text{Finance}))]) \\ & \bowtie [\pi_{D.*} \sigma_{F_1.budget > 150 \wedge D_2.did=F_1.did \wedge D_2.floor=D.floor} \\ & (\rho_D(\text{Dept}) \times \rho_{D_2}(\text{Dept}) \times \rho_{F_1}(\text{Finance}))]) \end{aligned}$$

Cost-based plan selection

Task

(refer to the handouts for the full exercise)

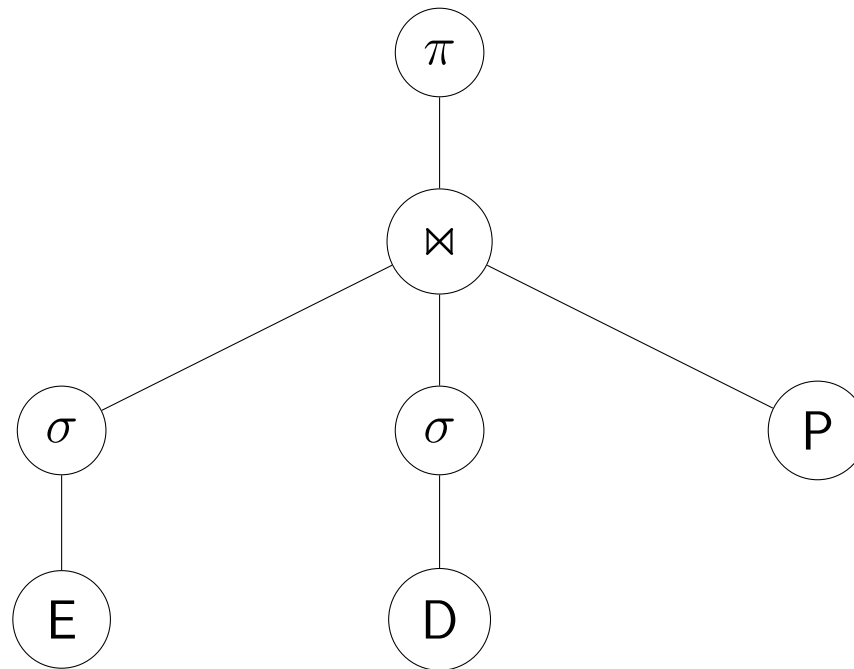
Construct a sufficiently optimal physical query plan for:

$$\pi_{E.eid,D.did,P.pid}\sigma_{E.sal=50000}(E) \bowtie \sigma_{D.budget \geq 20000}(D) \bowtie P$$

Assume that employee salaries are uniformly distributed over the range [10009, 110008] and that project budgets are uniformly distributed over [10000, 30000]. There are clustered indexes available on E.sal, D.did and P.pid.

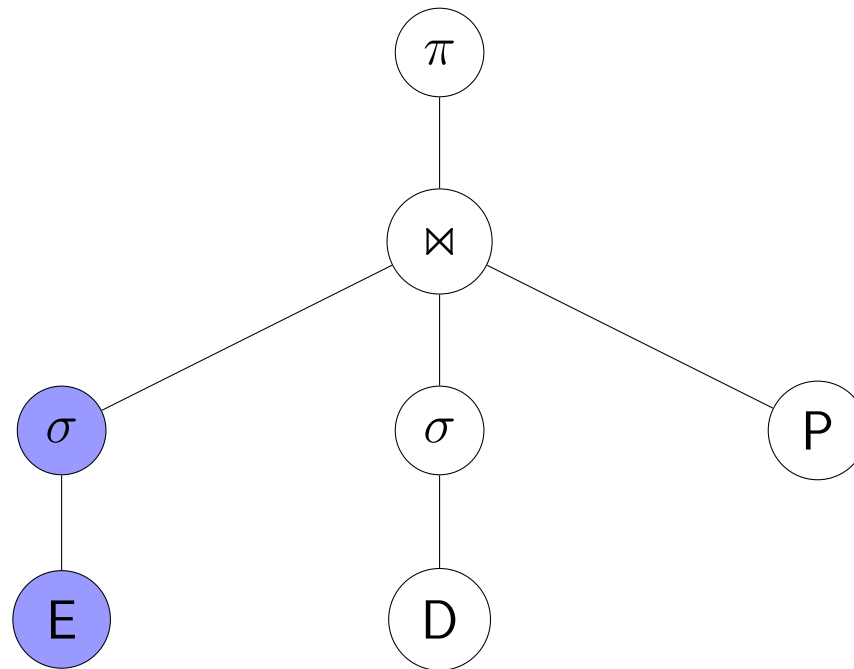
Cost-based plan selection

Solution



Cost-based plan selection

Solution



Cost-based plan selection

Solution

Subexpression:

$$\sigma_{E.sal=50000}(E)$$

First possibility: we use the clustered index on `E.sal` to get the records such that `E.sal = 50000`.

The number of tuples that satisfy the salary requirement is estimated to:

$$\left[\frac{1}{110008 - 10009} \text{ selectivity} \times 20000 \text{ employees} \right] = 1 \text{ tuples}$$

Hence, the result can be stored in 1 block:

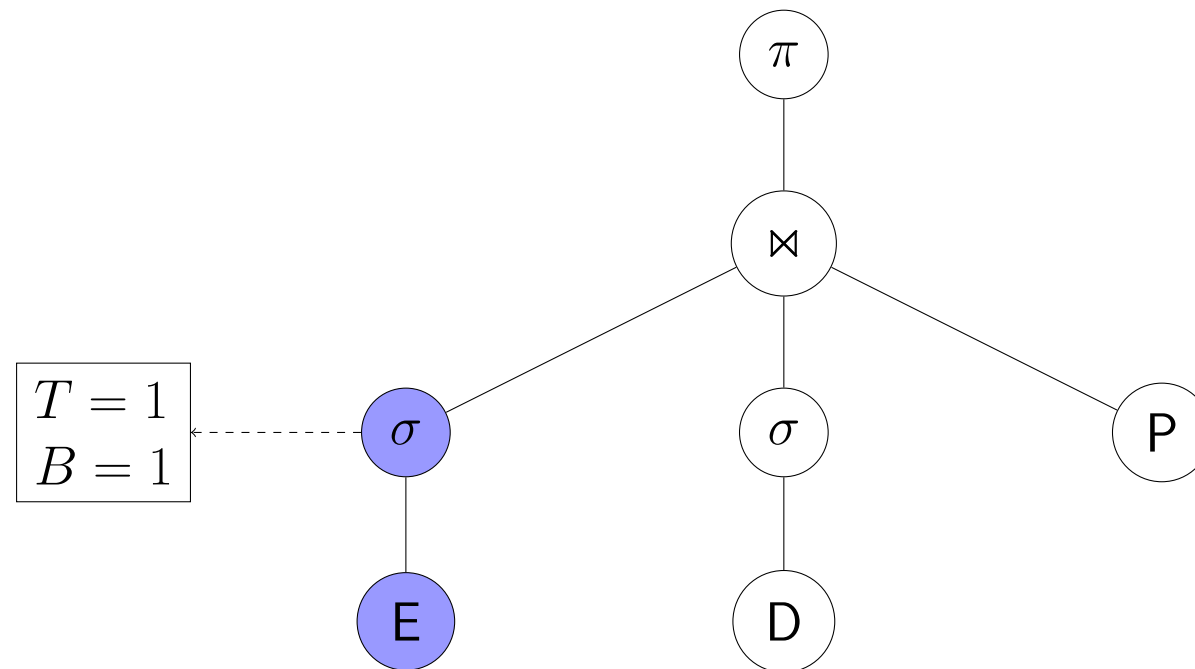
$$\frac{20 \text{ bytes}}{4000 \text{ bytes/block}} = 1 \text{ block}$$

A table scan would cost:

$$\frac{20 \text{ bytes/tuple} \times 20000}{4000 \text{ bytes/block}} = 100 \text{ block I/Os}$$

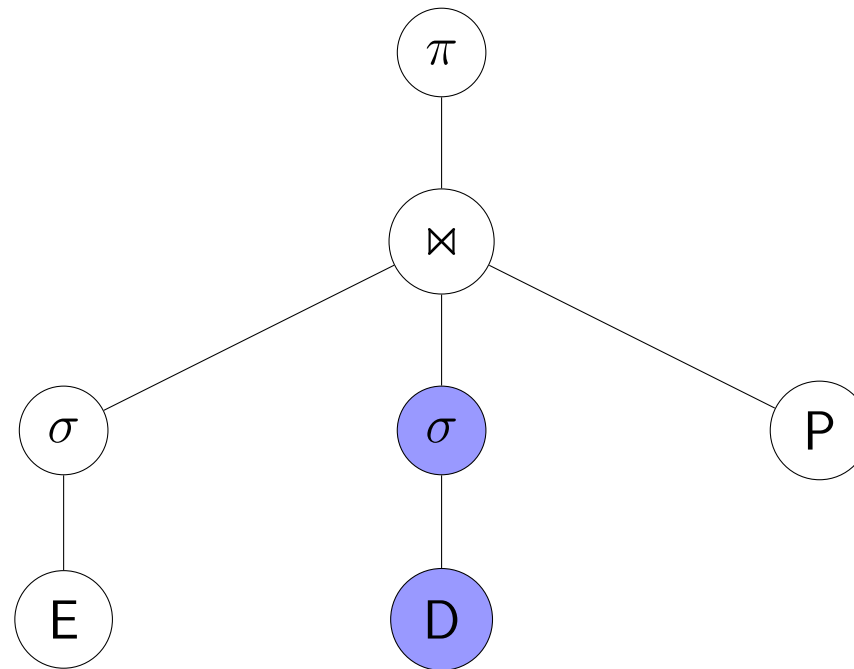
Cost-based plan selection

Solution



Cost-based plan selection

Solution



Cost-based plan selection

Solution

Subexpression:

$$\sigma_{D.\text{budget} \geq 20000}(D)$$

The number of tuples returned is estimated to 2500:

$$\left[\frac{30000 - 20000}{30000 - 10000} \text{selectivity} \times 5000 \text{ departments} \right] = 2500 \text{ tuples}$$

This corresponds to 25 Blocks:

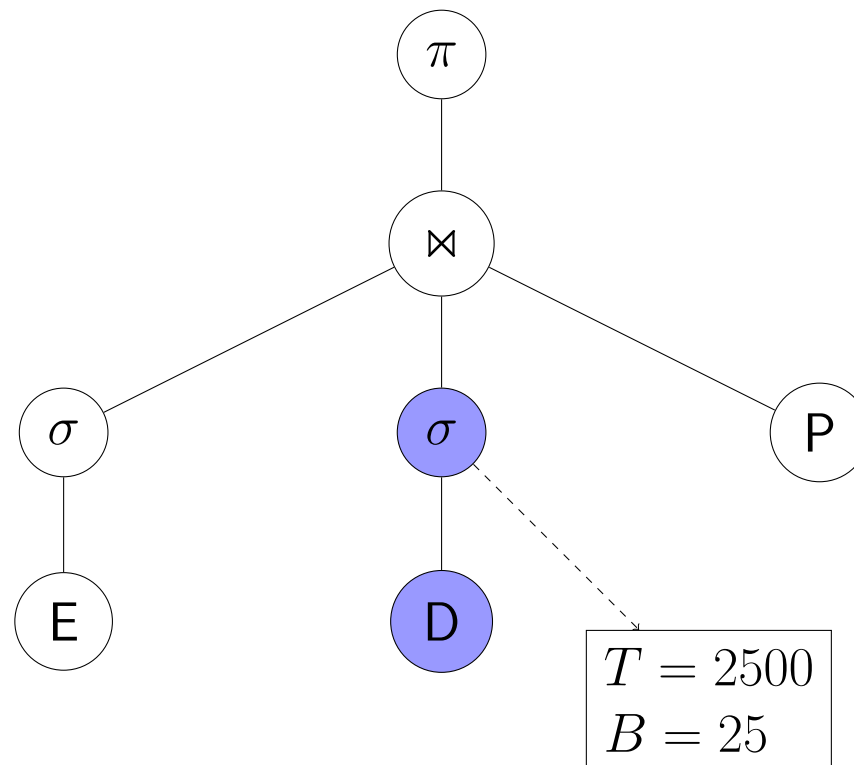
$$\frac{40 \text{ bytes/tuple} \times 2500}{4000 \text{ bytes/block}} = 25 \text{ blocks}$$

Since no index is available, a table scan is our only possibility:

$$\frac{40 \text{ bytes/tuple} \times 5000}{4000 \text{ bytes/block}} = 50 \text{ blocks}$$

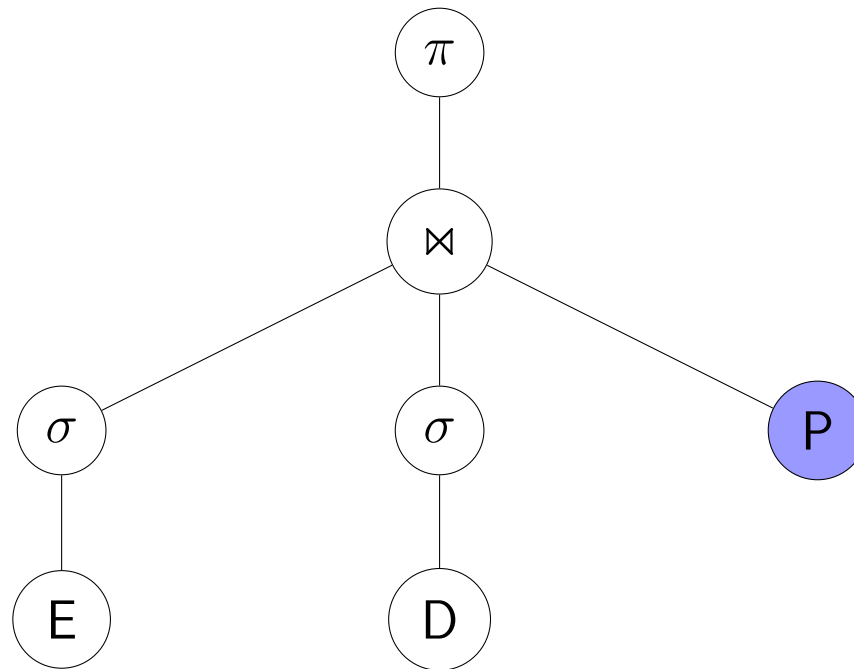
Cost-based plan selection

Solution



Cost-based plan selection

Solution



Cost-based plan selection

Solution

Subexpression:

P

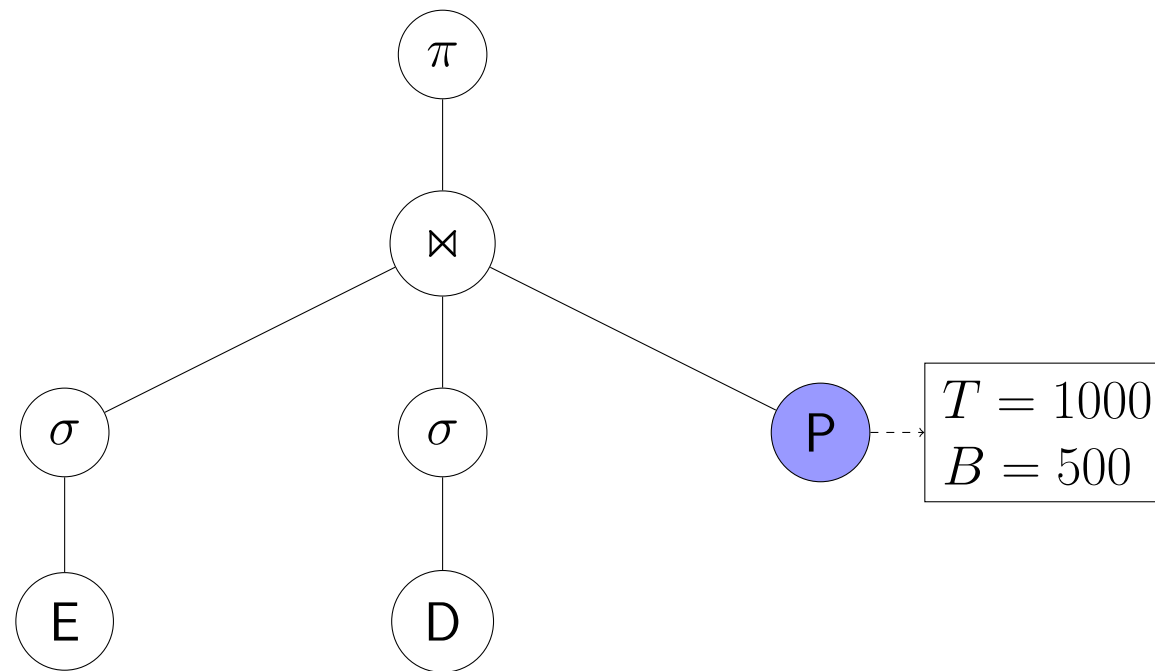
A table scan on P requires 500 block I/O's. This is also the estimated number of blocks returned:

$$\frac{2000 \text{ bytes/tuple} \times 1000 \text{ tuples}}{4000 \text{ bytes/block}} = 500 \text{ block}$$

.

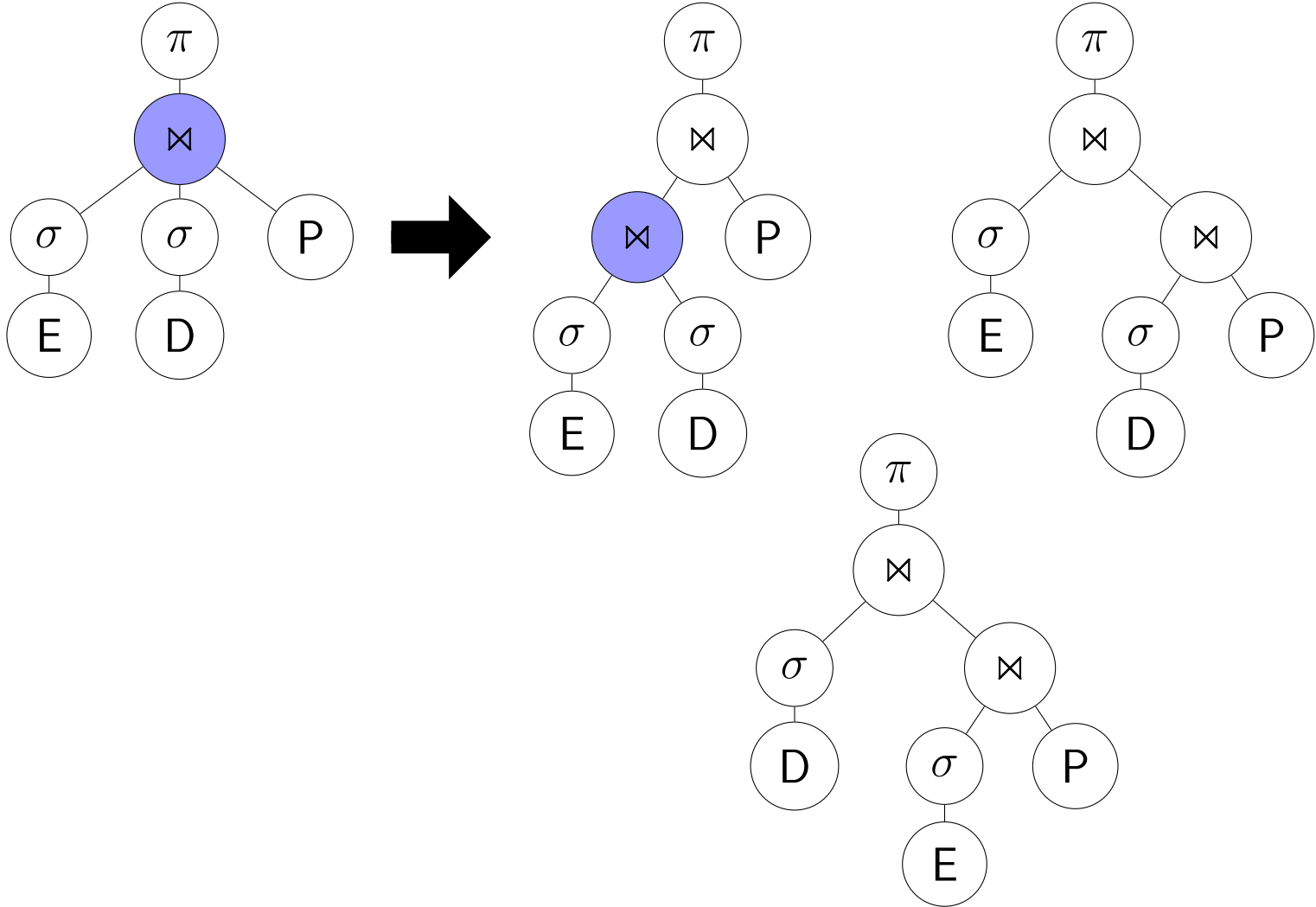
Cost-based plan selection

Solution



Cost-based plan selection

Solution



Cost-based plan selection

Solution

Now, we must determine an ordering for the joins. We consider all pairs of joins and keep the one with the smallest cost.

$$\underbrace{\sigma_{e.sal=50000}(E)}_{e_1} \text{ and } \underbrace{\sigma_{d.budget \geq 20000}(D)}_{e_2}$$

The selection on each side requires one buffer to execute, leaving only 10 buffers for the join.

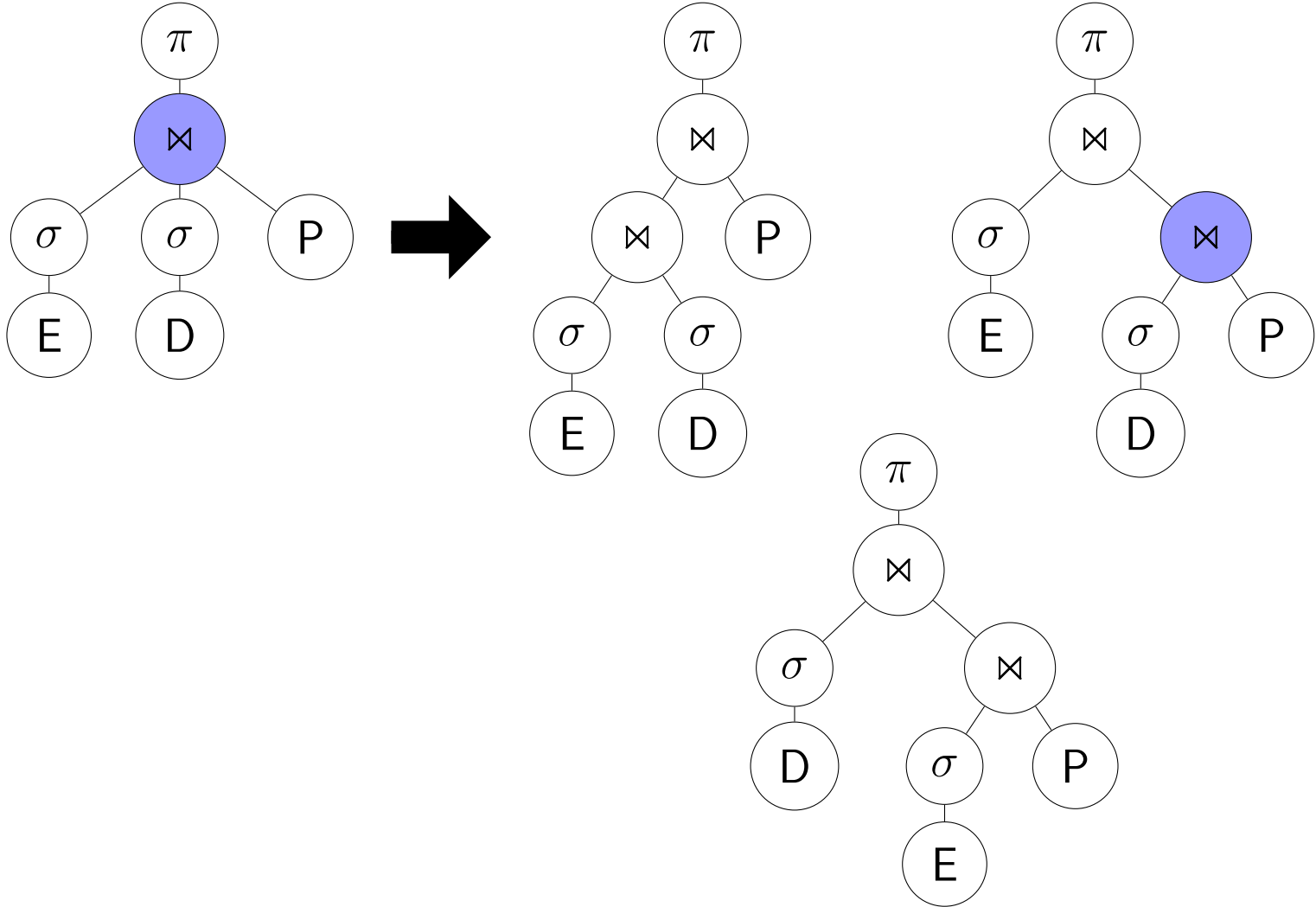
The output of e_1 contains only 1 tuples, and can therefore be computed in 1 block. Since $1 = B(e_1) \leq M = 10$, we can apply the one-pass join algorithm. Its cost is

$$B(e_1) + B(e_2) = 1 + 25 = 26 \text{ I/O's}$$

An index-join cannot be used on e_2 since it is not a base relation. All other join methods always cost more than one-pass join. Hence the one-pass join is preferred.

Cost-based plan selection

Solution



Cost-based plan selection

Solution

The second join pair is:

$$\underbrace{\sigma_{D.\text{budget} \geq 20000}(D)}_{e_2} \text{ and } P$$

We have 11 buffers at our disposal, given that we need 1 buffer to perform the selection in e_2 . It is not possible to use a one-pass join, since $25 = B(e_2) \geq M = 11$ and $500 = B(P) \geq M = 11$.

The non-optimized sort-merge join has a cost of:

$$\begin{aligned} & 2B(e_2) \lceil \log_M B(e_2) \rceil + 2B(P) \lceil \log_M B(P) \rceil + B(e_2) + B(P) \\ &= 2 \times 25 \times 2 + 2 \times 500 \times 3 + 25 + 500 \\ &= 3625 \text{ I/O's} \end{aligned}$$

We cannot use the optimization here, since there is not enough memory to perform the last merge of the merge-sort along with that of the sort-join:

$$49 \text{ necessary buffers} = \left\lceil \frac{B(e_2)}{M} \right\rceil + \left\lceil \frac{B(P)}{M} \right\rceil \not\leq M = 11 \text{ available buffers}$$

Cost-based plan selection

Solution

Assuming that the clustered index on $P.pid$ is a $BTree$, it ensues that P is already sorted on this join attribute. Given that we just have to sort e_2 , the cost is:

$$2B(e_2) \lceil \log_M B(e_2) \rceil + B(e_2) + B(P)$$

Futhermore, we can optimize the last merge:

$$4 \text{ necessary buffers} = \left\lceil \frac{B(e_2)}{M} \right\rceil + 1 \leq M = 11 \text{ available buffers}$$

The cost thereof is:

$$\begin{aligned} & 2B(e_2)(\lceil \log_M B(e_2) \rceil - 1) + B(e_2) + B(P) \\ &= 2 \times 25 \times 1 + 25 + 500 \\ &= 575 \text{ I/Os} \end{aligned}$$

Cost-based plan selection

Solution

The cost of an hash-join is:

$$\begin{aligned} & 2B(e_2) \lceil \log_{M-1} B(e_2) - 1 \rceil + 2B(P) \lceil \log_{M-1} B(e_2) - 1 \rceil + B(e_2) + B(P) \\ &= 2 \times 25 \times 1 + 2 \times 500 \times 1 + 25 + 500 \\ &= 1575 \text{ I/O's} \end{aligned}$$

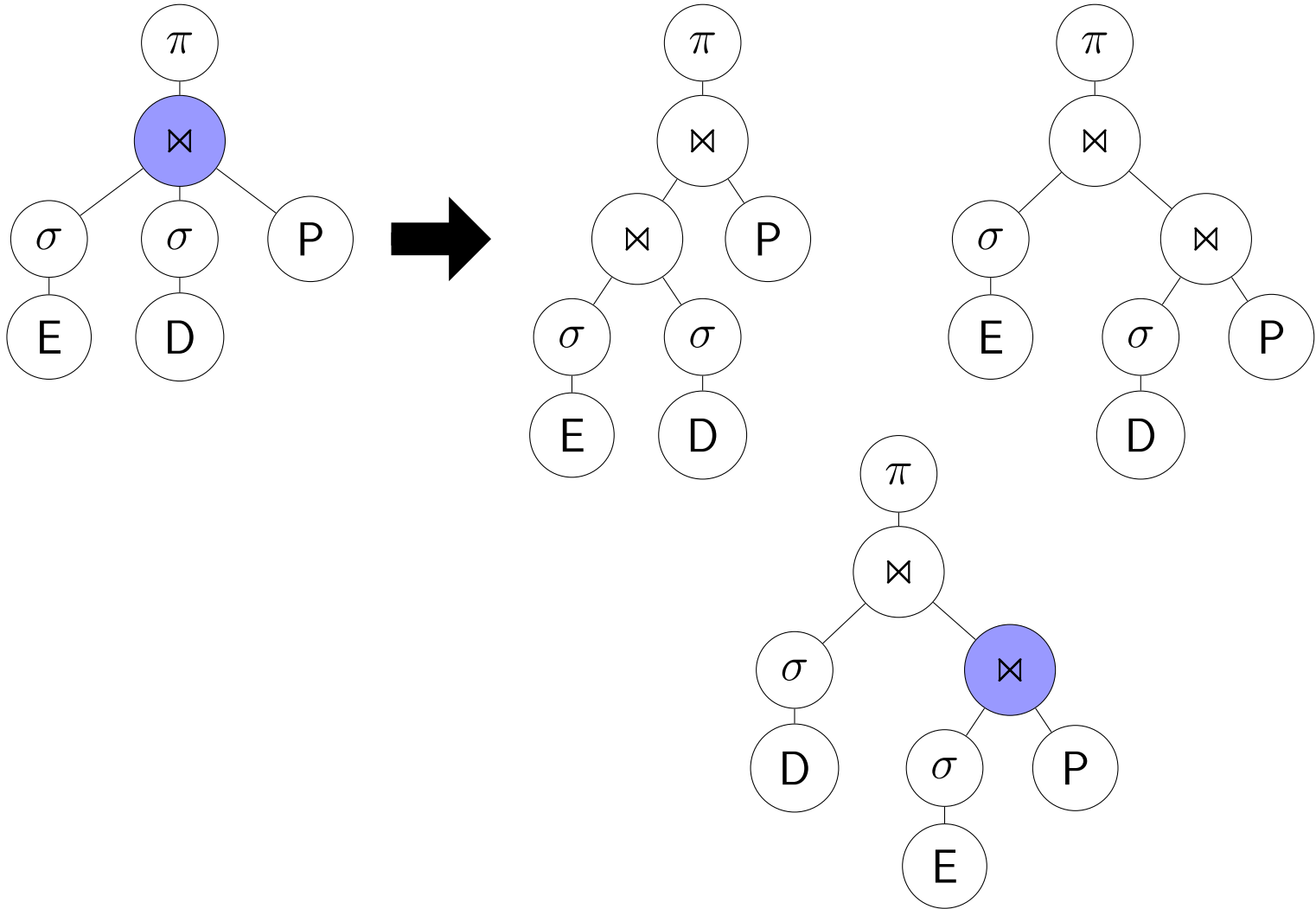
It is also possible to use an index-join, using the clustered index on P.did. This method has a cost of:

$$B(e_1) + T(e_1) \times \left\lceil \frac{V(P, \text{pid}) \times \text{Size}_{\text{ptuple}}}{\text{Size}_{\text{block}}} \right\rceil = 25 + 2500 \times 1 = 2525 \text{ I/O's}$$

Hence, the optimized sort-merge join (using the sorted index) is therefore preferred if the index on P.did is a BTree. The hash-join is preferred otherwise.

Cost-based plan selection

Solution



Cost-based plan selection

Solution

The third join pair is:

$$\underbrace{\sigma_{E.sal=50000}(E)}_{e_1} \text{ and } P$$

Note that this join is a full cartesian product. A one-pass join is available at the following cost:

$$B(e_1) + B(P) = 1 + 500 = 501 \text{ I/O's}$$

No index can help up for this join, and the one-pass join algorithm gives the best cost.

Cost-based plan selection

Solution

The join-pair with the least cost is therefore:

$$\underbrace{\sigma_{e.sal=50000}(E)}_{e_1} \text{ and } \underbrace{\sigma_{D.budget \geq 20000}(D)}_{e_2}$$

Where an one-pass join on E.did is used. Therefore, only 2 buffers are necessary (why?).

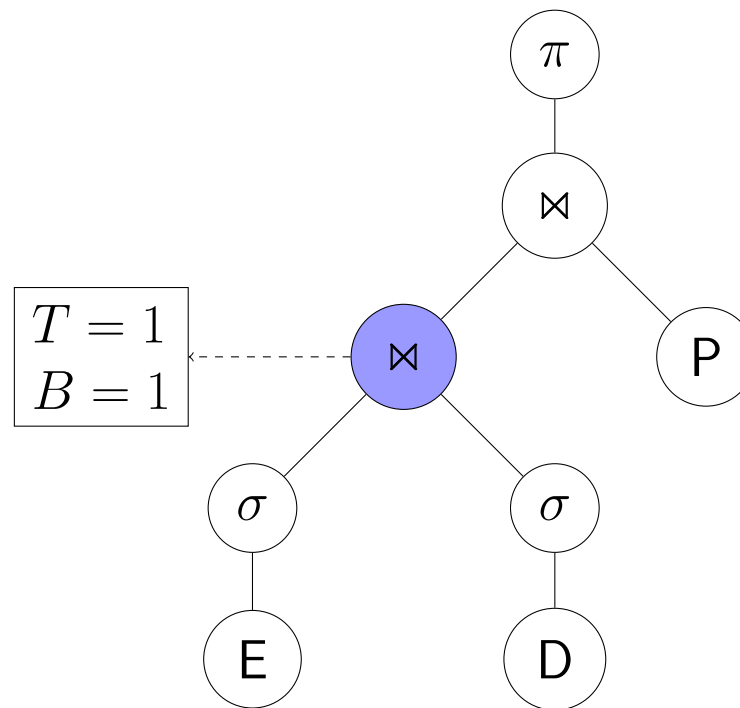
The estimated number of tuples in the output of this join is:

$$\frac{T(e_1) \times T(e_2)}{\max(V(e_1, did), V(e_2, did))} = \frac{1 \times 2500}{2500} = 1$$

These records are 60 bytes long and can be stored in 1 blocks

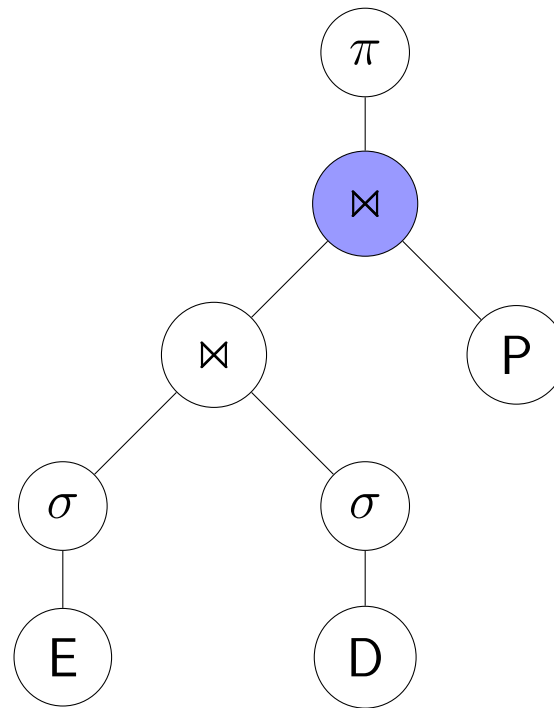
Cost-based plan selection

Solution



Cost-based plan selection

Solution



Cost-based plan selection

Solution

We still need to find the best way to join the whole expression

$$\underbrace{\sigma_{e.sal=50000}(E) \bowtie \sigma_{D.budget \geq 20000}(D)}_{e_3} \text{ and } P$$

The output of e_3 fits in 1 blocks. Given that $1 = B(e_3) \leq M = 12$, a one-pass join is possible. The cost thereof is:

$$B(e_3) + B(P) = 1 + 500 = 501$$

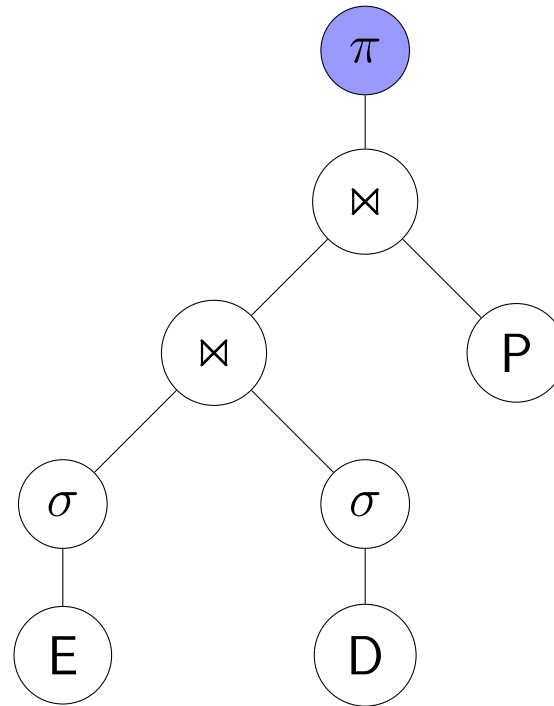
This joins can also be performed by means of an index-join, using the clustered index on $P.pid$.

$$B(e_3) + T(e_3) \times \left\lceil \frac{\text{Size}_{P\text{-tuple}}}{\text{Size}_{\text{block}} \times V(P, pid)} \right\rceil = 1 + 1 \times 1 = 1 \text{ I/O's}$$

Hence, the index-join is preferred.

Cost-based plan selection

Solution



Cost-based plan selection

Solution

The projection $\pi_{E.eid,D.did,P.pid}$ can be performed on the fly at the same time as the last join.

Notice that we did not need to materialize any of the intermediate results.

Cost-based plan selection

Solution

