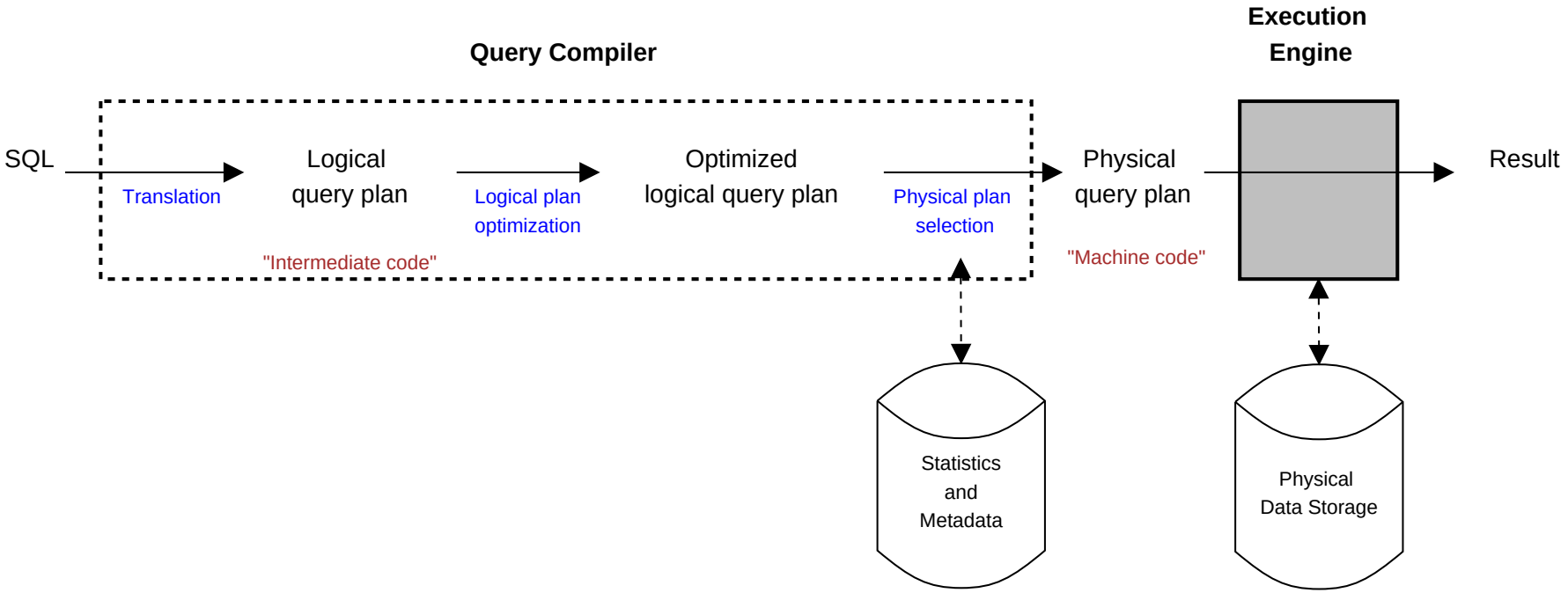
Cost-Based Plan Selection

Enumerate, Estimate, Select

Cost-Based Plan Selection

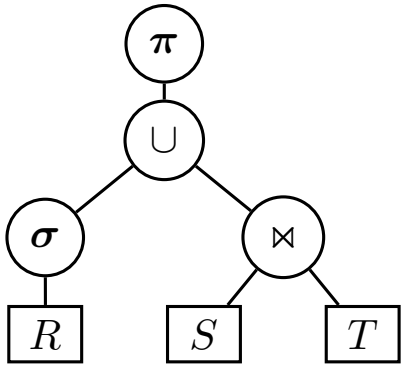


Components of the query compiler that we already know:

- SQL $\rightarrow$ relational algebra (i.e., a logical query plan)
- Logical query plan $\rightarrow$ optimized logical query plan

Cost-Based Plan Selection

The next step: logical query plan $\rightarrow$ physical query plan

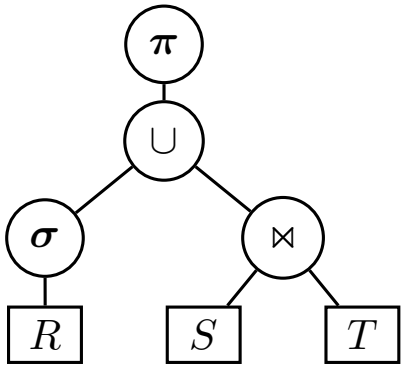


- To obtain a physical query plan we need to assign to each node in the logical query plan a physical operator.
- We want to obtain the physical plan with the smallest total execution cost.
- Hence, we need to compare, for every node and every applicable physical operator, its cost.

- In order to estimate this cost we need (among others) the parameters $B(R)$, $T(R)$, and $V(R, A_1, \ldots, A_n)$
- These belong to the statistics that a DBMS typically stores in its system catalog
- But these statistics only exist for the relations stored in the database, not for subresults computed during query evaluation!

Cost-Based Plan Selection

Result size estimation



- For every internal node n we hence need to estimate the parameters $B(n)$, $T(n)$, and $V(n, A_1, \ldots, A_k)$
- Note that we can compute $B(n)$ given (1) $T(n)$; (2) the size of the tuples output by n; and (3) the size of a block
- Also note that $T(n)$ and $V(n, A_1, \ldots, A_k)$ only depend on the logical query plan, not on the physical plan that we are computing!

Cost-Based Plan Selection

Result size estimation: projection

- General formula: $T(\pi_L(R)) = T(R)$
- Remember that our version of the projection operator is bag-based and does not remove duplicates; to remove duplicates we use the operator δ.
- While projection does not change the number of tuples, it does change the number of blocks needed to store the resulting relation, as illustrated by the following example.

Example

- $R(A, B, C)$ is a relation with A and B integers of 4 bytes each; C a string of 100 bytes. Tuple headers are 12 bytes. Blocks are 1024 bytes and have headers of 24 bytes. $T(R) = 10000$ and $B(R) = 1250$.
- Question: how many blocks do we need to store $\pi_{A,B}(R)$?

Cost-Based Plan Selection

Result size estimation: projection

- General formula: $T(\pi_L(R)) = T(R)$
- Remember that our version of the projection operator is bag-based and does not remove duplicates; to remove duplicates we use the operator δ.
- While projection does not change the number of tuples, it does change the number of blocks needed to store the resulting relation, as illustrated by the following example.

Example

- $R(A, B, C)$ is a relation with A and B integers of 4 bytes each; C a string of 100 bytes. Tuple headers are 12 bytes. Blocks are 1024 bytes and have headers of 24 bytes. $T(R) = 10000$ and $B(R) = 1250$.
- **Answer:** resulting records need to record the header + A-field + B-field. The size of these records is hence $12 + 4 + 4 = 20$ bytes. We can hence store $(1024 - 24)/20 = 50$ tuples in one block. Thus $B(\pi_{A,B}(R)) = T(\pi_{A,B}(R))/50 = 10000/50 = 200$ blocks.

Cost-Based Plan Selection

Result size estimation: selection $\sigma_P(R)$ with P a filter predicate

- General formula:

$$\boxed{T(\sigma_P(R)) = T(R) \times \mathit{sel}_P(R)}$$

  where $\mathit{sel}_P(R)$ is the estimated fraction of tuples in R that satisfy predicate P.

- In other words, $\mathit{sel}_P(R)$ is the estimated probability that a tuple in R satisfies P.
- $\mathit{sel}_P(R)$ is usually called the selectivity of filter predicate P.
- How we calculate $\mathit{sel}_P(R)$ depends on what P is.

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{A=c}(R)$ with c a constant

$$\boxed{sel_{A=c}(R) = \frac{1}{V(R,A)}}$$

- Intuition: there are $V(R, A)$ distinct A-values in R. Assuming that A-values are uniformly distributed, the probability that a tuple has A-value c is $1/V(R, A)$.
- While this intuition assumes that values are uniformly distributed, it can be shown that this selectivity is a good estimate on average, provided that c is chosen randomly.

Example

- $R(A, B, C)$ is a relation. $T(R) = 10000$. $V(R, A) = 50$.
- Then $T(\sigma_{A=10}(R))$ is estimated by:

$$T(\sigma_{A=10}(R)) = T(R) \times \frac{1}{V(R, A)} = \frac{10000}{50} = 200.$$

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{A=c}(R)$ with c a constant

- Better selectivity estimates are possible if we have more detailed statistics
- A DBMS typically collects histograms that detail the distribution of values.
- Such histograms are only available for base relations, however, not for subresults!

Example

- $R(A, B, C)$ is a relation. The DBMS has collected the following equal-width histogram on A:

range	$[1, 10]$	$[11, 20]$	$[21, 30]$	$[31, 40]$	$[41, 50]$
tuples in range	50	2000	2000	3000	2950

- Then $\mathit{sel}_{A=10}(R)$ can be estimated by:

$$\mathit{sel}_{A=10}(R) = \frac{50}{10000} \times \frac{1}{10}$$

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{A<c}(R)$

$$\boxed{sel_{A<c}(R) = \frac{1}{2}} \quad \text{or} \quad \boxed{sel_{A<c}(R) = \frac{1}{3}}$$

- This is just a heuristic, without any correctness guarantees.
- (The intuitive rationale is that queries involving an inequality tend to retrieve a small fraction of the possible tuples.)

Example

- $R(A, B, C)$ is a relation. $T(R) = 10000$.
- Then $T(\sigma_{B<10}(R))$ is estimated by:

$$T(\sigma_{B<10}(R)) = T(R) \times \frac{1}{3} = 3334.$$

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{A<c}(R)$

- Again, better estimates are possible if we have more detailed statistics

Example

- $R(A, B, C)$ is a relation. $T(R) = 10000$. The DBMS statistics show that the values of the B attribute lie within the range $[8, 57]$, uniformly distributed.
- Question: what would be a reasonable estimate of $sel_{B<10}(R)$?

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{A<c}(R)$

- Again, better estimates are possible if we have more detailed statistics

Example

- $R(A, B, C)$ is a relation. $T(R) = 10000$. The DBMS statistics show that the values of the B attribute lie within the range $[8, 57]$, uniformly distributed.
- Question: what would be a reasonable estimate of $\mathit{sel}_{B<10}(R)$?
- Answer: We see that $57 - 8 + 1$ different values of B are possible; however only records with values $B = 8$ or $B = 9$ satisfy the filter $B < 10$. Therefore,

$$\mathit{sel}_{B<10}(R) = \frac{2}{(57 - 8 + 1)} = \frac{2}{50} = 4\%$$

and hence

$$T(\sigma_{B<10}(R)) = T(R) \times \mathit{sel}_{B<10}(R) = 400.$$

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{A \neq c}(R)$

$$\boxed{sel_{A \neq c}(R) = \frac{V(R,A)-1}{V(R,A)}}$$

- Question: Can you give intuitive meaning to this formula?

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{A \neq c}(R)$

$$\boxed{sel_{A \neq c}(R) = \frac{V(R,A)-1}{V(R,A)}}$$

- Question: Can you give intuitive meaning to this formula?
- Answer: $1/V(R, A)$ is the (estimated) probability that a tuple satisfies $A = c$. Therefore

$$1 - sel_{A=c}(R) = 1 - \frac{1}{V(R, A)} = \frac{V(R, A) - 1}{V(R, A)}$$

is the (estimated) probability that a tuple does not satisfy $A = c$.

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{\textsf{NOT}\ P_1}(R)$

$$\boxed{sel_{\textsf{NOT}\ P_1}(R) = 1 - sel_{P_1}(R)}$$

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{P_1 \text{ AND } P_2}(R)$

$$\boxed{\mathit{sel}_{P_1 \text{ AND } P_2}(R) = \mathit{sel}_{P_1}(R) \times \mathit{sel}_{P_2}(R)}$$

- This implicitly assumes that filter predicates P_1 and P_2 are independent.
- Hence, in essence we treat $\sigma_{P_1 \text{ AND } P_2}(R)$ as $\sigma_{P_1}(\sigma_{P_2}(R))$
- The order does not matter, treating this as $\sigma_{P_2}(\sigma_{P_1}(R))$ gives the same results.

Example

- $R(A, B, C)$ is a relation. $T(R) = 10000$. $V(R, A) = 50$.
- Then we estimate $T(\sigma_{A=10 \text{ AND } B<10}(R)$ to be:

$$T(R) \times \mathit{sel}_{A=10}(R) \times \mathit{sel}_{B<10}(R) = T(R) \times \frac{1}{V(R, A)} \times \frac{1}{3} = 67.$$

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{P_1 \text{ OR } P_2}(R)$

$$\boxed{sel_{P_1 \text{ OR } P_2}(R) = \min(sel_{P_1}(R) + sel_{P_2}(R), 1)}$$

- The term $sel_{P_1}(R) + sel_{P_2}(R)$ implicitly assumes that filter predicates P_1 and P_2 are independent, and select disjoint sets of tuples.
- Disjointness is often not satisfied and then we count some tuples twice.
- But of course, the selectivity can never be greater than 1.
- Hence, we take the minimum of these two terms.

Cost-Based Plan Selection

Result size estimation: selection $\sigma_{P_1 \text{ OR } P_2}(R)$

More complicated: treat this as $\sigma_{\text{NOT}(\text{NOT } P_1 \text{ AND NOT } P_2)}(R))$.

$$\boxed{\mathit{sel}_{P_1 \text{ OR } P_2}(R) = 1 - (1 - \mathit{sel}_{P_1}(R)) \times (1 - \mathit{sel}_{P_2}(R))}$$

Cost-Based Plan Selection

Result size estimation: cartesian product $R \times S$

- General formula:

$$\boxed{T(R \times S) = T(R) \times T(S)}$$

Cost-Based Plan Selection

Result size estimation: natural join $R \bowtie S$

- Assume the relation schema $R(X, Y)$ and $S(Y, Z)$, i.e., we join on Y.
- Many cases are possible
  - It is possible that R and S do not have any Y value in common. In that case, $T(R \bowtie S) = 0$.
  - Y might be the key of S and a foreign key of R, so each tuple of R joins with exactly one tuple of S. Then $T(R \bowtie S) = T(R)$.
  - Almost all of the tuples of R and S could have the same Y-value. Then $T(R \bowtie S)$ is approximately $T(R) \times T(S)$.

Cost-Based Plan Selection

Result size estimation: natural join $R \bowtie S$

- Assume the relation schema $R(X, Y)$ and $S(Y, Z)$, i.e., we join on Y.
- To focus on the common cases, we make two simplifying assumptions.
  1. Containment of value sets If attribute Y appears in several relations, then each relation chooses its values from a fixed list of values $y_1, y_2, y_3, \ldots$. As a consequence, if $V(R, Y) \leq V(S, Y)$ then every Y-value of R will have a joining tuple Y-value in S.
  2. Preservation of value sets When joining two relations, any attribute that is not a join attribute does not lose values from its set of possible values: for such attributes $V(R \bowtie S, A) = V(R, A)$, when A is in R and $V(R \bowtie S, A) = V(S, A)$ otherwise.

Cost-Based Plan Selection

Result size estimation: natural join $R \bowtie S$

- Assume the relation schema $R(X, Y)$ and $S(Y, Z)$, i.e., we join on Y.
- Under these assumptions, we can estimate as follows.
  1. **Case 1:** $V(R, Y) \leq V(S, Y)$. Then every tuple of R has $\frac{1}{V(S,Y)}$ chance of joining with a given tuple of S. Hence
  $$T(R \bowtie S) = T(R) \times \frac{1}{V(S, Y)} \times T(S)$$
  2. **Case 2:** $V(S, Y) \leq V(R, Y)$. Then every tuple of S has $\frac{1}{V(R,Y)}$ chance of joining with a given tuple of R. Hence
  $$T(R \bowtie S) = T(R) \times \frac{1}{V(R, Y)} \times T(S)$$

Cost-Based Plan Selection

Result size estimation: natural join $R \bowtie S$

- Assume the relation schema $R(X, Y)$ and $S(Y, Z)$, i.e., we join on Y.
- Under these assumptions, we can estimate as follows.
  1. **Case 1:** $V(R, Y) \leq V(S, Y)$. Then every tuple of R has $\frac{1}{V(S,Y)}$ chance of joining with a given tuple of S. Hence

$$T(R \bowtie S) = T(R) \times \frac{1}{V(S, Y)} \times T(S)$$

  2. **Case 2:** $V(S, Y) \leq V(R, Y)$. Then every tuple of S has $\frac{1}{V(R,Y)}$ chance of joining with a given tuple of R. Hence

$$T(R \bowtie S) = T(R) \times \frac{1}{V(R, Y)} \times T(S)$$

General formula:

$$\boxed{T(R \bowtie S) = T(R) \times T(S) \times \frac{1}{\max(V(R,Y), V(S,Y))}}$$

Cost-Based Plan Selection

Result size estimation: natural join $R \bowtie S$

- Now assume the relation schema $R(X, Y_1, Y_2)$ and $S(Y_1, Y_2, Z)$, i.e., we join on Y_1 and Y_2.
- Under the same assumptions as before, we can estimate as follows.

  Case 1: $V(R, Y_1) \leq V(S, Y_1)$ and $V(R, Y_2) \leq V(S, Y_2)$.

  Then a tuple of R has $\frac{1}{V(S,Y_1)} \times \frac{1}{V(S,Y_2)}$ chance of joining with a given tuple of S. Hence

$$T(R \bowtie S) = T(R) \times \frac{1}{V(S, Y_1)} \times \frac{1}{V(S, Y_2)} \times T(S)$$

Cost-Based Plan Selection

Result size estimation: natural join $R \bowtie S$

- Now assume the relation schema $R(X, Y_1, Y_2)$ and $S(Y_1, Y_2, Z)$, i.e., we join on Y_1 and Y_2.
- Under the same assumptions as before, we can estimate as follows.

  Case 2: $V(S, Y_1) \leq V(R, Y_1)$ and $V(S, Y_2) \leq V(R, Y_2)$.

  Then a tuple of S has $\frac{1}{V(R,Y_1)} \times \frac{1}{V(R,Y_2)}$ chance of joining with a given tuple of R. Hence

$$T(R \bowtie S) = T(R) \times \frac{1}{V(R, Y_1)} \times \frac{1}{V(R, Y_2)} \times T(S)$$

Cost-Based Plan Selection

Result size estimation: natural join $R \bowtie S$

- Now assume the relation schema $R(X, Y_1, Y_2)$ and $S(Y_1, Y_2, Z)$, i.e., we join on Y_1 and Y_2.
- Under the same assumptions as before, we can estimate as follows.

  Case 3: $V(R, Y_1) \leq V(S, Y_1)$ and $V(S, Y_2) \leq V(R, Y_2)$.

  Then a tuple of R has $\frac{1}{V(S,Y_1)} \times \frac{1}{V(R,Y_2)}$ chance of joining with a given tuple of S. Hence

$$T(R \bowtie S) = T(R) \times \frac{1}{V(S, Y_1)} \times \frac{1}{V(R, Y_2)} \times T(S)$$

Cost-Based Plan Selection

Result size estimation: natural join $R \bowtie S$

- Now assume the relation schema $R(X, Y_1, Y_2)$ and $S(Y_1, Y_2, Z)$, i.e., we join on Y_1 and Y_2.
- Under the same assumptions as before, we can estimate as follows.

  Case 4: $V(S, Y_1) \leq V(R, Y_1)$ and $V(R, Y_2) \leq V(S, Y_2)$.

  Then a tuple of R has $\frac{1}{V(R,Y_1)} \times \frac{1}{V(S,Y_2)}$ chance of joining with a given tuple of S. Hence

$$T(R \bowtie S) = T(R) \times \frac{1}{V(R, Y_1)} \times \frac{1}{V(S, Y_2)} \times T(S)$$

Cost-Based Plan Selection

Result size estimation: natural join $R \bowtie S$

- Now assume the relation schema $R(X, Y_1, Y_2)$ and $S(Y_1, Y_2, Z)$, i.e., we join on Y_1 and Y_2.

- General formula:

$$\boxed{T(R \bowtie S) = \frac{T(R) \times T(S)}{\max(V(R,Y_1),V(S,Y_1)) \max(V(R,Y_2),V(S,Y_2))}}$$

- This generalizes straightforwardly to the case where we are joining on more than 2 attributes.

Cost-Based Plan Selection

Result size estimation

- Intersection $R \cap S$, Difference $R - S$, duplicate elimination $\delta(R)$, Grouping and aggregation $\gamma(R)$

  $\rightarrow$ see section 16.4 in the book

- A DBMS often also collects more detailed statistics

  $\rightarrow$ see sections 16.5.1 and 16.5.2 in the book

- As should be clear by now, result size estimation is not an exact art

- For commercial DBMSs, the software component that estimates result sizes is intricate and advanced!

Cost-Based Plan Selection

Join ordering

During the optimization of the logical query plan we:

- remove redundant joins;
- push selections and projections; recognize joins.

The order in which the joins are to be executed is not yet fixed, however!

Cost-Based Plan Selection

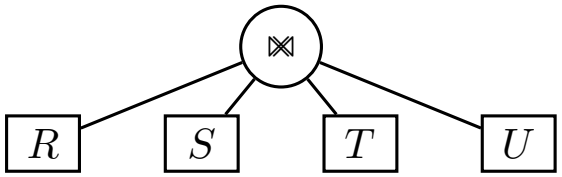
Join ordering

Example: relations $R(A,B), S(B,C), T(C,D), U(D,A)$ and the query

SQL:
```
SELECT * FROM R,S,T,U
WHERE R.B = S.B AND S.C = T.C AND T.D = U.D
```

Algebra: $R \bowtie S \bowtie T \bowtie U$

- So far, we have always considered the join as a polyadic operator:



  After all, the join order is irrelevant for logical query plans.
- However, physical join operators are binary!
- When devising a physical query plan, the join order therefore becomes very important, as we illustrate next.

Cost-Based Plan Selection

Join ordering

Example: relations $R(A,B), S(B,C), T(C,D), U(D,A)$ and the query

SQL:
```
SELECT * FROM R,S,T,U
WHERE R.B = S.B AND S.C = T.C AND T.D = U.D
```

Algebra: $R \bowtie S \bowtie T \bowtie U$

We can interpret this as:

$$((R \bowtie S) \bowtie T) \bowtie U \quad \text{or} \quad (R \bowtie S) \bowtie (T \bowtie U) \quad \text{or} \quad \ldots$$

But also as:

$$((R \bowtie T) \bowtie U) \bowtie S \quad \text{or} \quad ((R \bowtie S) \bowtie U) \bowtie T \quad \text{of} \quad \ldots$$

Cost-Based Plan Selection

Join ordering

The chosen order can influence the total cost of the physical query plan.

Consider, for example, $R(A,B), S(B,C), T(A,E)$. Assume

$$B(R) = 50 \qquad B(S) = 50 \qquad B(T) = 50$$
$$B(R \bowtie S) = 150 \qquad B(S \bowtie T) = 2500 \qquad B(R \bowtie T) = 200$$

Further assume that we execute all joins by means of the one-pass algorithm. What is the best order to compute $R \bowtie S \bowtie T$?

1. **Cost of** $R \bowtie (S \bowtie T)$**:**

$$B(R) + B(S \bowtie T) + B(S) + B(T) = 2650$$

2. **Cost of** $S \bowtie (R \bowtie T)$**:**

$$B(S) + B(R \bowtie T) + B(R) + B(T) = 350$$

3. **Cost of** $T \bowtie (R \bowtie S)$**:**

$$B(T) + B(R \bowtie S) + B(R) + B(S) = 300$$

Cost-Based Plan Selection

Join ordering

- To obtain the physical plan with the least cost we would hence have to enumerate and compare every possible join ordering.
- The number of possible orderings to join n relations is $n! \times T(n)$:
  - There are $n!$ ways to order the relations to join
  - Given a fixed ordering, there are $T(n)$ ways to create a binary tree over n leaf nodes, where

$$T(1) = 1 \qquad\qquad T(n) = \sum_{i=1}^{n-1} T(i) \times T(n-i)$$

Cost-Based Plan Selection

Join ordering

- The resulting search space is enormous.

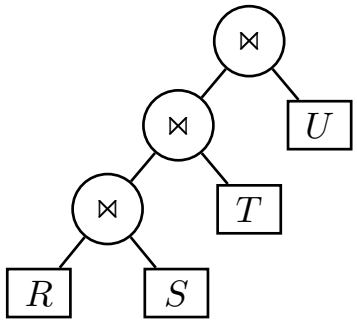
Number of relations n	$n! \times T(n)$
2	2
3	12
4	120
5	1,680
6	30,240
7	665,580
8	17,297,280

- For each of these plans, we have to consider all possible assignments of physical join algorithms to logical join operators to get the plan with the least cost.

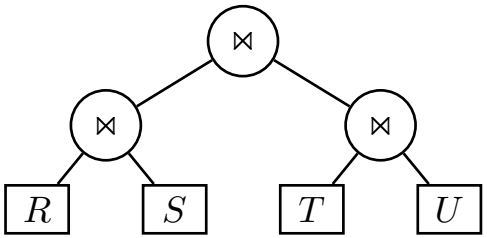
  $\rightarrow$ Query optimization should in no case take more time than the actual execution of the query. We will therefore not consider all possible orders, but only a limited subclass.

Cost-Based Plan Selection

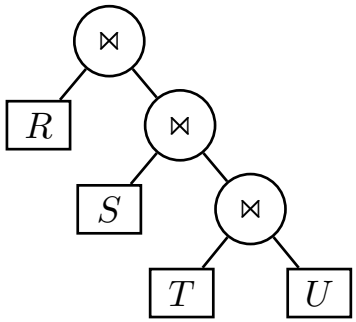
Kinds of join orderings



left-deep



bushy



right-deep

In practice a query compiler usually only considers left-deep join orderings:

- There are still $n!$ possible orderings of this form, but that is already a lot less.
- Left-deep orderings use, in general, less memory. Furthermore, in general they require fewer subresults to be stored.

→ See section 16.6.3 in the book

Cost-Based Plan Selection

Plan selection

To compute the best physical plan for a given logical query plan we should, in principle:

1. Calculate all possible (left-deep) join orderings of the logical plan
2. For each such plan calculate all possible assignments of physical operators to the nodes
3. From this enormous pile of candidate physical query plans choose the one with the least estimated cost.

There are exponentially many candidate physical query plans

- Query compilation should in no case take longer than the actual execution of the query!
- In general it is hence impossible to inspect all candidate physical plans.

  Heuristics: Branch-and-Bound Plan Enumeration; Hill Climbing; Dynamic Programming; Selinger-Style; Greedy

  → See section 16.5.4, 16.6.4 and 16.6.5 in the book

Cost-Based Plan Selection

Greedy plan selection

In the exercises we will use the following greedy algorithm.

- Start with a logical query plan without join ordering.
- We work bottom-up: first we assign physical operators to the leaves, then to the parents of the leaves, then to their parents, and so on. At each point we choose the phyiscal operator with the least cost.
- When we reach a join operator (e.g., $R \bowtie S \bowtie T \bowtie U$) and need to determine an ordering of its various members then:
  1. We start by joining the two relations for which the best physical join algorithm yields the smallest cost
     $\rightarrow$ e.g., execute $R \bowtie T$ through a hash-join
  2. Add, from the remaining relations (S or U), those relations to the join for which the best physical join-algorithm yields the smallest cost.
     $\rightarrow$ e.g., $(R \bowtie T) \bowtie U$ through a one-pass join
  3. Repeat the previous step until we have a complete join ordering.

Cost-Based Plan Selection

Greedy plan selection

- This is a generalization of the greedy algorithm to compute a join ordering described in section 16.6.6 from the book. However, we use I/O operations as our cost metric instead of the size of the intermediate results as done in the book.
- Often, the leaves of the logical query plan are selections. We have seen two physical operators for selections: table-scan and index-scan. The book describes in section 16.7.1 how we can choose the best selection method when the selection condition is complex.

Cost-Based Plan Selection

Greedy plan selection need not return the optimal plan

- It may return a more expensive join ordering. For example:

$$R(A,B) \bowtie S(B,C) \bowtie T(C,D) \bowtie U(A,D)$$

Assume: the greedy algorithm computes $((R \bowtie S) \bowtie T) \bowtie U)$ with

$$B(R \bowtie S) = 100 \qquad B((R \bowtie S) \bowtie T) = 2000$$

Assume: the alternative ordering $((R \bowtie U) \bowtie T) \bowtie S$ yields

$$B(R \bowtie U) = 200 \qquad B((R \bowtie U) \bowtie T) = 1000$$

When we hence execute the joins using the one-pass algorithm we get the following costs, respectively:

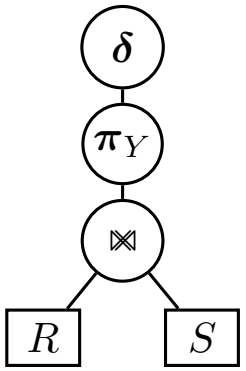
1. $B(R) + B(S) + B(R \bowtie S) + B(T) + B((R \bowtie S) \bowtie T) + B(U)$
2. $B(R) + B(U) + B(R \bowtie U) + B(T) + B((R \bowtie U) \bowtie T) + B(S)$

The second ordering yields a saving of 900 I/Os.

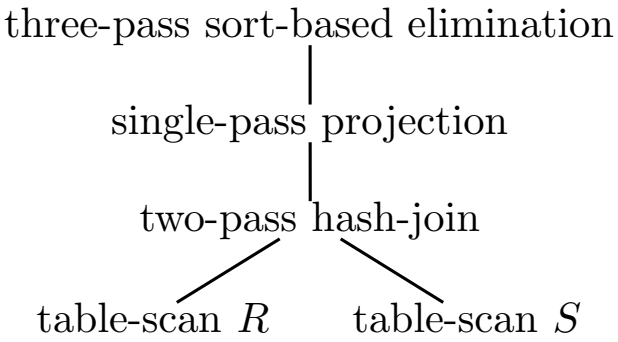
Cost-Based Plan Selection

Greedy plan selection need not return the optimal plan

- It does not take into account the properties of the output of an operator. For example (R and S share only the Y attribute):



logical plan



physical plan

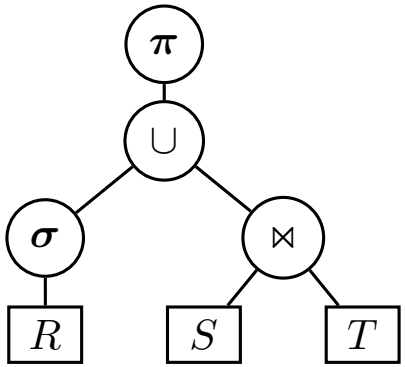
Consider the setting where there is limited memory available. The optimized sort-merge join is not applicable; only the non-optimized version. In this case the two-pass hash-join is cheaper, and is hence selected by the greedy algorithm. Because the output of $R \bowtie S$ is large, we will eventually have to remove duplicates by means of a three-pass algorithm.

If, however, we had executed the join by means of a two-pass sort-merge join, then its result would have been sorted on Y and we would have been able to compute the duplicate removal by means of the one-pass algorithm instead of the three-pass one. In that case, the total costs would have been smaller (check this!)

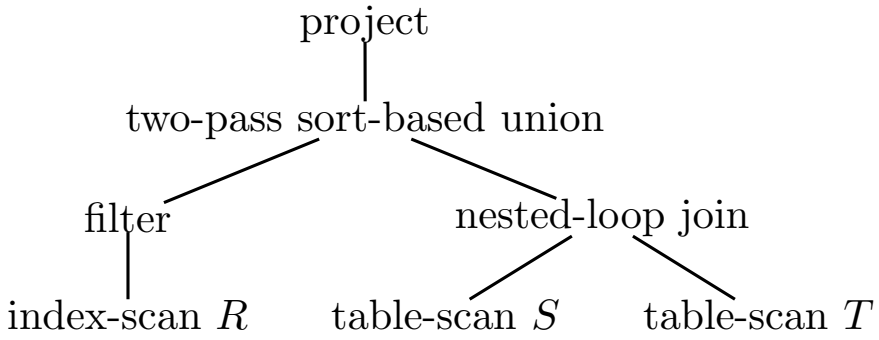
Cost-Based Plan Selection

Finally

The result of the greedy algorithm is an execution tree in which every node is a physical operator.



logical plan



physical plan

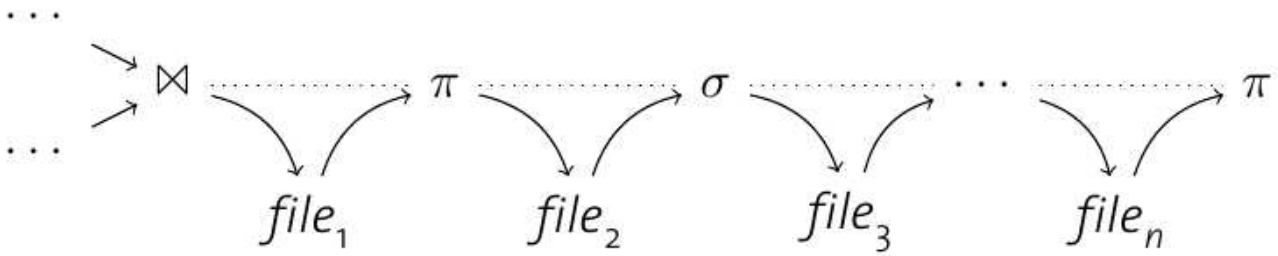
We remain to decide, for every internal node, whether we will materialize or pipeline the subresults.

→ See sections 16.7.3, 16.7.4, and 16.7.5

Cost-Based Plan Selection

Pipelining versus materialization

So far, we have assumed that all database operators consume items on disk, and produce their result on disk.



- This causes a lot of I/O.
- In addition, we suffer from long response times since an operator cannot start computing its result before **all** of its inputs are fully generated ("materialized")

Cost-Based Plan Selection

Pipelining versus materialization

Alternatively, each operator could pass its result directly to the next operator. This is called pipelining.

When executed in a pipelined manner, an operator

- Starts computing results as early as possible, i.e., as soon as enough input data is available to start producing output.
- Doesn't wait until the entire output is computed, but propagates its output immediately.

The granularity in which data is passed may influence performance:

- Small chunks yield better system response time.
- Large chunks may improve the effectiveness of caches.
- Most often, data is passed a tuple at a time.

Cost-Based Plan Selection

Examples of operators that can be pipelined

- projection
- selection
- renaming
- bag-based union
- merge-joins for which the input are already known to be sorted

Cost-Based Plan Selection

Pipelining versus materialization

Pipelining reduces memory requirements and response times since each chunk of its input is propagated to the output immediately.

Some operators cannot be implemented in such a way:

- operators based on (external) sorting (i.e. sort-merge join)
- operators based on external hashing (i.e., hash join)
- grouping and duplicate elimination over unsorted input

Operators that cannot be pipelined are said to be blocking

- Blocking operators consume their entire input before they can produce any output.
- Their data is typically materialized on disk.