Chapter 9: Query Optimization

Zhaonian Zou

Massive Data Computing Research Center
School of Computer Science and Technology
Harbin Institute of Technology, China
Email: znzou@hit.edu.cn

Spring 2019

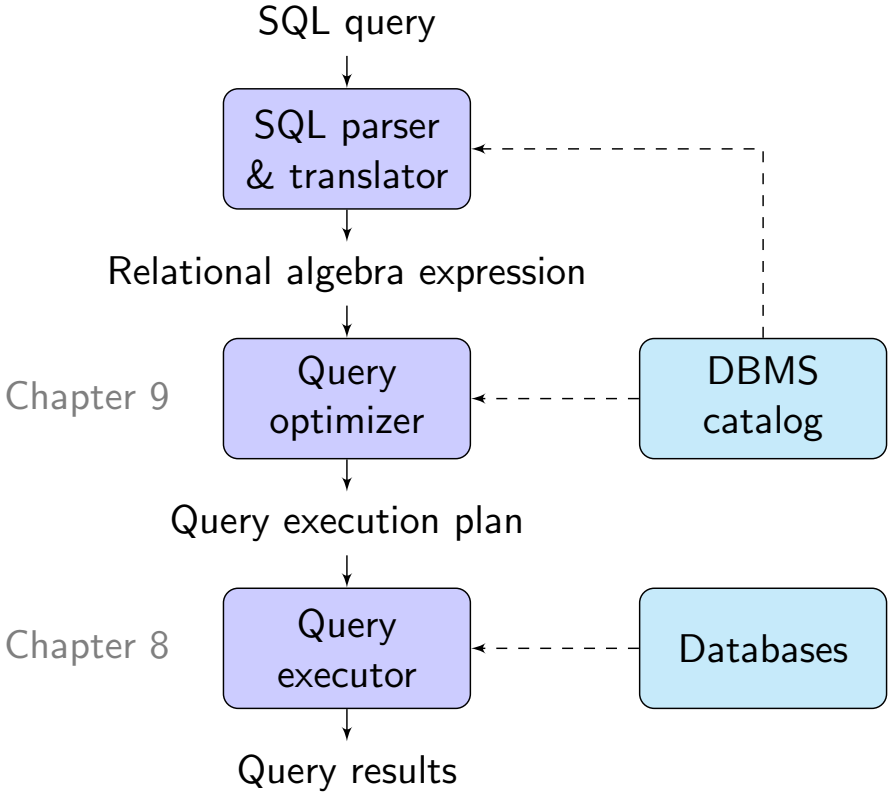
Outline[1]

1. 9.1 Overview of Query Optimization
2. 9.2 Improving Logical Query Plans
   - 9.2.1 Transformations of Relational-Algebra Expressions
   - 9.2.2 Estimating the Cost of Operations
   - 9.2.3 Choosing an Order for Joins
3. 9.3 Improving Physical Query Plans
   - 9.3.1 Selection of Algorithms for Operators
   - 9.3.2 Materialization vs. Pipelining

[1] Updated on April 10, 2019

9.1 Overview of Query Optimization

Overview of Query Processing (查询处理)



Query Optimization (查询优化)

- Logical query optimization (逻辑查询优化) transforms an initial logical query plan (逻辑查询计划) represented using relational algebra to a logical query plan with lower estimated cost
- Physical query optimization (物理查询优化) generates a physical query plan (物理查询计划), according to which the query can be evaluated by the DBMS more efficiently

Example (Query Optimization)

- SQL query:

```
SELECT ID, salary FROM instructor WHERE salary < 75000;
```

Initial logical plan	Alternative logical plan	Physical query plan
$\Pi_{ID,salary}$	$\sigma_{salary<75000}$	$\Pi_{ID,salary}$
|	|	|
$\sigma_{salary<75000}$	$\Pi_{ID,salary}$	$\sigma_{salary<75000}$; use index
|	|	|
instructor	instructor	instructor

Costs of Query Plans

In a disk-based DBMS, the cost of evaluating a query is approximated well by the number of disk I/O's performed, which in turn is influenced by:

1. The logical operators chosen to implement the query
2. The sizes of intermediate relations
3. The ordering of similar operations
4. The physical operators used to implement logical operators
5. The method of passing arguments from one physical operator to the next

- Logical query optimization considers factors 1–3 (Section 9.2)
- Physical query optimization considers factors 4–5 (Section 9.3)

9.2 Improving Logical Query Plans

Improving Logical Query Plans

- Select the logical operators to implement the query (Section 9.2.1)
- Estimate the sizes of intermediate relations (Section 9.2.2)
- Choose the orders of similar operations (Section 9.2.3)

9.2 Improving Logical Query Plans
9.2.1 Transformations of Relational-Algebra Expressions

Transformations of Relational-Algebra Expressions

Equivalent Relational-Algebra Expressions (等价关系代数表达式)

Two relational-algebra expressions are equivalent if they yield the same result on every legal instance of an relational database

Example (Equivalent Expressions)

Initial expression

$\Pi_{ID,salary}$

|

$\sigma_{salary<75000}$

|

instructor

Alternative expression

$\sigma_{salary<75000}$

|

$\Pi_{ID,salary}$

|

instructor

Goal: Turn one expression tree into an equivalent expression tree that may have a more efficient physical query plan

Laws of Transformations (等价变换规则)

Several of the operators of relational algebra are both commutative (满足交换律) and associative (满足结合律)

- $R \times S = S \times R$; $(R \times S) \times T = R \times (S \times T)$
- $R \bowtie S = S \bowtie R$; $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- $R \cap S = S \cap R$; $(R \cap S) \cap T = R \cap (S \cap T)$
- $R \cup S = S \cup R$; $(R \cup S) \cup T = R \cup (S \cup T)$

Theta-join is commutative but not associative

- $R \bowtie_\theta S = S \bowtie_\theta R$
- $(R \bowtie_{\theta_1} S) \bowtie_{\theta_2} T \neq R \bowtie_{\theta_1} (S \bowtie_{\theta_2} T)$; e.g., $(R(a,b) \bowtie_{R.b>S.b} S(b,c)) \bowtie_{a<d} T(c,d)$

Laws involving Selections

The splitting laws

- $\sigma_{\theta_1 \wedge \theta_2}(R) = \sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_2}(\sigma_{\theta_1}(R))$
- $\sigma_{\theta_1 \vee \theta_2}(R) = \sigma_{\theta_1}(R) \cup \sigma_{\theta_2}(R)$

Pushing a selection through a union

- $\sigma_\theta(R \cup S) = \sigma_\theta(R) \cup \sigma_\theta(S)$

Pushing a selection through a difference

- $\sigma_\theta(R - S) = \sigma_\theta(R) - S = \sigma_\theta(R) - \sigma_\theta(S)$

Pushing a selection through an intersection

- $\sigma_\theta(R \cap S) = \sigma_\theta(R) \cap S = R \cap \sigma_\theta(S) = \sigma_\theta(R) \cap \sigma_\theta(S)$

Laws involving Selections (Cont'd)

Pushing a selection through a product

- $\sigma_\theta(R \times S) = R \bowtie_\theta S$
- $\sigma_\theta(R \times S) = \sigma_\theta(R) \times S$ if R has all the attributes mentioned in θ, but S has not

Pushing a selection through a theta-join

- $\sigma_{\theta_1}(R \bowtie_{\theta_2} S) = R \bowtie_{\theta_1 \wedge \theta_2} S$
- $\sigma_{\theta_1}(R \bowtie_{\theta_2} S) = \sigma_{\theta_1}(R) \bowtie_{\theta_2} S$ if R has all the attributes mentioned in θ, but S has not

Pushing a selection through a natural join

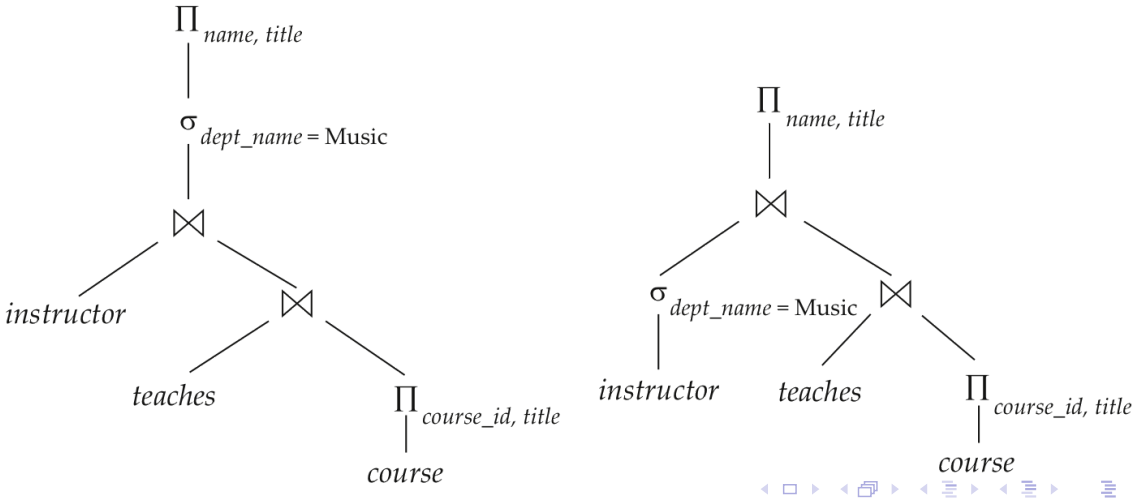
- $\sigma_\theta(R \bowtie S) = \sigma_\theta(R) \bowtie S$ if R has all the attributes mentioned in θ, but S has not
- $\sigma_\theta(R \bowtie S) = \sigma_\theta(R) \bowtie S = R \bowtie \sigma_\theta(S) = \sigma_\theta(R) \bowtie \sigma_\theta(S)$ if both R and S have all the attributes mentioned in θ

Laws involving Selections (Cont'd)

Example (Pushing Selections)

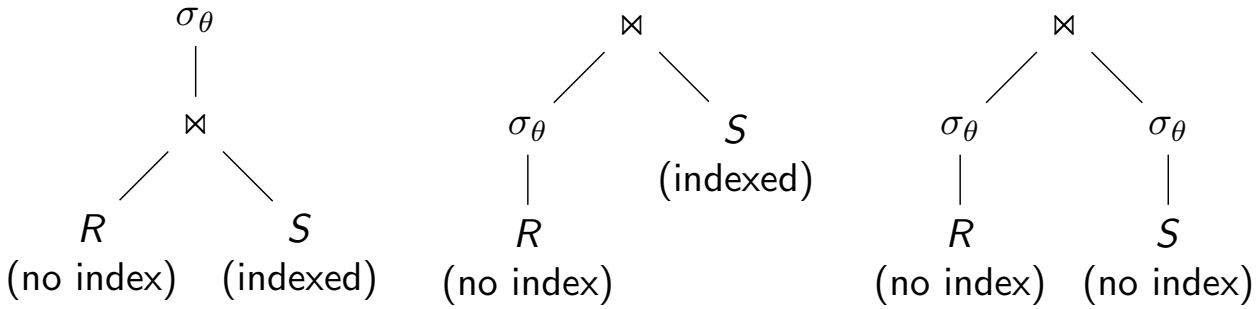
- instructor(ID, name, dept_name, salary)
- teaches(ID, course_id, sec_id, semester, year)
- course(course_id, title, dept_name, credits)

$\Pi_{name,title}(\sigma_{dept\_name="Music"}(instructor \bowtie teaches \bowtie \Pi_{course\_id,title}(course)))$



Heuristics of Query Optimization (1)

- We often push a selection down an expression tree (选择下推)
- When it is possible to push a selection to both arguments of a binary operator, we need to decide whether or not to do so. How would the existence of indexes on one of the arguments affect our choice?



Initial expression | Good transformation | Bad transformation

- Sometimes, we first move a selection as far up the tree as it would go before we push the selections down all possible branches

$$\Pi_{name,title}(\sigma_{dept\_name="Music"}(instructor) \bowtie teaches \bowtie course)$$

Laws involving Projections

Pushing a projection through a projection (the absorbing law)

- $\Pi_{L_1}(\Pi_{L_2}(R)) = \Pi_{L_1}(R)$

Pushing a projection through a selection

- $\Pi_L(\sigma_\theta(R)) = \Pi_L(\sigma_\theta(\Pi_M(R)))$, where M is the list of all attributes that are either input attributes of L or mentioned in condition θ

Pushing a projection through a product

- $\Pi_L(R \times S) = \Pi_L(\Pi_M(R) \times \Pi_N(S))$, where M and N are the lists of all attributes of R and S, respectively, that are input attributes of L

Laws involving Projections (Cont'd)

Pushing a projection through a natural join

- $\Pi_L(R \bowtie S) = \Pi_L(\Pi_M(R) \bowtie \Pi_N(S))$, where M and N are the lists of all attributes of R and S, respectively, that are either join attributes or are input attributes of L

Pushing a projection through a theta-join

- $\Pi_L(R \bowtie_\theta S) = \Pi_L(\Pi_M(R) \bowtie_\theta \Pi_N(S))$, where M and N are the lists of all attributes of R and S, respectively, that are either join attributes or are input attributes of L

Pushing a projection through a union

- $\Pi_L(R \cup S) = \Pi_L(R) \cup \Pi_L(S)$

Laws involving Projections (Cont'd)

Projections cannot be pushed below set intersections and set differences

- $\Pi_L(R \cap S) \neq \Pi_L(R) \cap \Pi_L(S)$
- $\Pi_L(R - S) \neq \Pi_L(R) - \Pi_L(S)$

Example

- $R(a, b) = \{(1, 3)\}$
- $S(a, b) = \{(1, 2)\}$
- $\Pi_a(R \cap S) = \emptyset$
- $\Pi_a(R) \cap \Pi_a(S) = \{(1)\}$
- $\Pi_a(R - S) = \{(1)\}$
- $\Pi_a(R) - \Pi_a(S) = \emptyset$

Laws involving Projections (Cont'd)

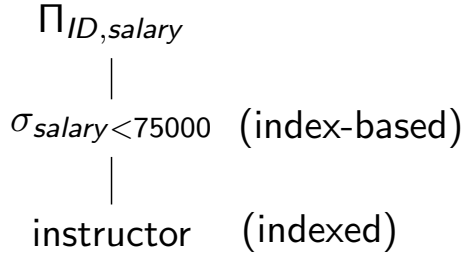
Example (Pushing Projections)

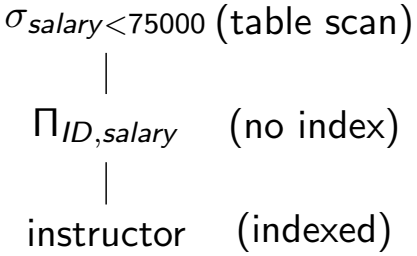
- instructor(ID, name, dept_name, salary)
- teaches(ID, course_id, sec_id, semester, year)
- course(course_id, title, dept_name, credits)

$$\Pi_{name,title}(\sigma_{dept\_name="Music"}(instructor \bowtie teaches \bowtie \Pi_{course\_id,title}(course)))$$
$$=\Pi_{name,title}(\sigma_{dept\_name="Music"}(\Pi_{ID,name,dept\_name}(instructor)) \bowtie \Pi_{ID,course\_id}(teaches) \bowtie \Pi_{course\_id,title}(course))$$

Heuristics of Query Optimization (2)

- Often, we wish to push projections down expression trees (投影下推)
- However, there are some examples where pushing a projection down costs time
  - For example, there is an index on attribute *ID* for relation *instructor*; however, there is no index on the result of $\Pi_{ID,name,dept\_name}(instructor)$





9.2 Improving Logical Query Plans
9.2.2 Estimating the Cost of Operations

Estimating the Cost of Operations (代价估计)

- The physical query plan is selected to minimize the estimated cost (disk I/O's) of evaluating the query
- The size of intermediate relations is the simplest measure to estimate the number of disk I/O's

Requirements

- Accurate
- Easy to compute
- Logically consistent

Notation

- $T(R)$: the number of tuples of relation R
- $V(R, A)$: the number of distinct values of attribute(s) A in relation R

Estimating the Size of a Product

- $T(R \times S) = T(S \times R) = T(R)T(S)$
- $T(R_1 \times R_2 \times \cdots \times R_n) = T(R_1)T(R_2)\cdots T(R_n)$

Estimating the Size of a Projection

- $T(\Pi_L(R)) = T(R)$ (Projection without duplication elimination)
- $T(\Pi_L(R)) = V(R, L)$ (Projection with duplication elimination)

Estimating the Size of a Selection

$S = \sigma_{A=c}(R)$

- $T(S) = T(R)/V(R, A)$

$S = \sigma_{A\neq c}(R)$

- $T(S) = T(R)$
- $T(S) = T(R) - T(R)/V(R, A)$

$S = \sigma_{A>c}(R)$ or $S = \sigma_{A<c}(R)$

- $T(S) = T(R)/3$ (Database System Implementation, 2nd Ed.)
- $T(S) = T(R)/2$ (Database System Concepts, 6th Ed.)

Estimating the Size of a Selection (Cont'd)

$S = \sigma_{\theta_1\wedge\theta_2}(R)$

- $S = \sigma_{\theta_1\wedge\theta_2}(R) = \sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_2}(\sigma_{\theta_2}(R))$
- $T(S) = T(R)f_1f_2$, where $f_i = 1/V(R, A)$ if θ_i is of the form $A = c$; $f_i = 1 - 1/V(R, A)$ if θ_i is of the form $A \neq c$; and $f_i = 1/3$ if θ_i is of the form $A < c$ or $A > c$

$S = \sigma_{\theta_1\vee\theta_2}(R)$

- $S = \sigma_{\theta_1}(R) \cup \sigma_{\theta_2}(R) = R - \sigma_{\neg\theta_1\wedge\neg\theta_2}(R)$
- $T(S) = T(R)(1 - (1 - f_1)(1 - f_2))$

$S = \sigma_{\neg\theta}(R)$

- $S = R - \sigma_{\theta}(R)$
- $T(S) = T(R) - T(\sigma_{\theta}(R))$

Estimating the Size of a Natural Join with a Single Join Attribute

We first study $R(X, Y) \bowtie S(Y, Z)$, where Y is a single attribute

Assumptions

- Containment of Value Sets: If $V(R, Y) \leq V(S, Y)$, then every Y-value of R will be a Y-value of S
- Preservation of Value Sets: If A is an attribute of R but not of S, then $V(R \bowtie S, A) = V(R, A)$

Estimation Method

- $T(R \bowtie S) = T(R)T(S)/V(S, Y)$ if $V(R, Y) \leq V(S, Y)$
- $T(R \bowtie S) = T(R)T(S)/V(R, Y)$ if $V(S, Y) \leq V(R, Y)$
- $T(R \bowtie S) = T(R)T(S)/\max(V(R, Y), V(S, Y))$

Estimating the Size of a Natural Join with a Single Join Attribute (Cont'd)

Example

- $R(a, b) : T(R) = 1000, V(R, b) = 20$
- $S(b, c) : T(S) = 2000, V(S, b) = 50, V(S, c) = 100$
- $U(c, d) : T(U) = 5000, V(U, c) = 500$
- $T((R \bowtie S) \bowtie U) =?$
- $T(R \bowtie (S \bowtie U)) =?$

Estimating the Size of a Natural Join with Multiple Join Attributes

Consider $R(W, X, Y) \bowtie S(X, Y, Z)$, where X and Y are two attributes

Estimating $T(R \bowtie S)$

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

Example

- $R(a, b, c) : T(R) = 1000, V(R, b) = 20, V(R, c) = 100$
- $S(b, c, d) : T(S) = 2000, V(S, b) = 50, V(S, c) = 50$
- $T(R \bowtie S) =?$

Estimating the Size of a Natural Join with Multiple Join Attributes (Cont'd)

Example

- $R(a, b) : T(R) = 1000, V(R, b) = 20$
- $S(b, c) : T(S) = 2000, V(S, b) = 50, V(S, c) = 100$
- $U(c, d) : T(U) = 5000, V(U, c) = 500$
- $T((R \bowtie S) \bowtie U) = 400,000$
- $T(R \bowtie (S \bowtie U)) = 400,000$
- $T((R \bowtie U) \bowtie S) =?$

Estimating the Size of a Natural Join of Multiple Relations

Estimating $T(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)$

Let $S = R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$

$$T(S) = \frac{T(R_1)T(R_2)\cdots T(R_n)}{\prod_{A \in \{\text{attributes appearing in more than two relations}\}} V(A)}$$

Let $R_{i_1}, R_{i_2}, \ldots R_{i_n}$ be all relations containing attribute A and $V(R_{i_1}, A) \leq V(R_{i_2}, A) \leq \cdots \leq V(R_{i_n}, A)$

$$V(A) = V(R_{i_2}, A)V(R_{i_3}, A)\cdots V(R_{i_n}, A)$$

Properties

- No matter how we group and order $R_1, R_2, \cdots, R_n$, the estimation rules yield the same estimate for the size of the result.
- This estimate is the same that we get if we apply the rule for the join of all $R_1, R_2, \cdots, R_n$ as a whole

Estimating the Size of a Natural Join of Multiple Relations (Cont'd)

Example

- $R(a, b, c) : T(R) = 1000, V(R, b) = 20, V(R, c) = 200$
- $S(b, c, d) : T(S) = 2000, V(S, b) = 50, V(S, c) = 100, V(S, d) = 400$
- $U(b, e) : T(U) = 5000, V(U, b) = 200, V(U, e) = 500$
- $T((R \bowtie S) \bowtie U) =?$
- $T(R \bowtie (S \bowtie U)) =?$
- $T((U \bowtie R) \bowtie S) =?$
- $T(R \bowtie S \bowtie U) =?$

Estimating the Size of a Set Operation

Estimating $T(R \cup S)$

- Property: $\max(T(R), T(S)) \leq T(R \cup S) \leq T(R) + T(S)$
- Rule: $T(R \cup S) = \frac{T(R)+T(S)+\max(T(R),T(S))}{2}$

Estimating $T(R - S)$

- Property: $T(R) - T(S) \leq T(R - S) \leq T(R)$
- Rule: $T(R - S) = T(R) - T(S)/2$

Estimating $T(R \cap S)$

- Property: $0 \leq T(R \cap S) \leq \min(T(R), T(S))$
- Rule 1: $T(R \cap S) = \min(T(R), T(S))/2$
- Rule 2: $T(R \cap S) = T(R \bowtie S)$ because $R \cap S = R \bowtie S$

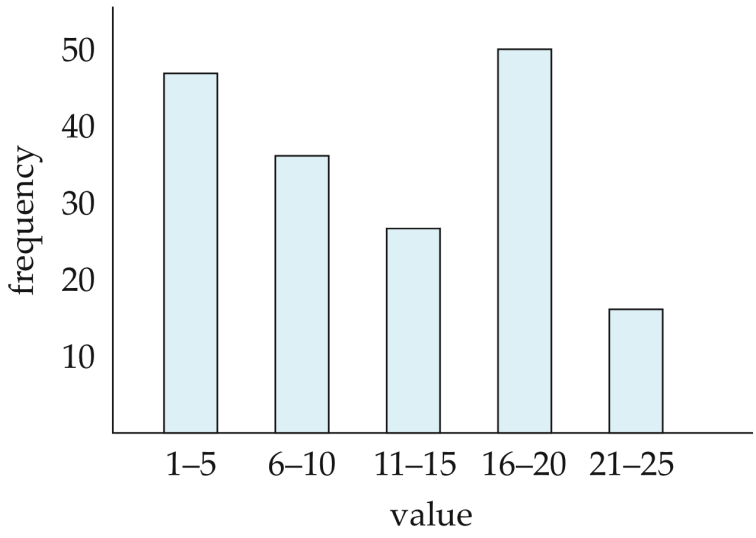
Estimating the Size of a Duplicate Elimination

Let $S = \delta(R(a_1, a_2, \ldots, a_n))$

- Property 1: $1 \leq T(S) \leq T(R)$
- Rule 1: $T(S) = (T(R) + 1)/2 \approx T(R)/2$
- Property 2: $1 \leq T(S) \leq V(R, a_1)V(R, a_2) \cdots V(R, a_n)$
- Rule 2: $T(S) = \min((T(R) + 1)/2, V(R, a_1)V(R, a_2) \cdots V(R, a_n))$

Histograms (直方图)

- If $V(R, A)$ is not too large, the histogram for attribute A may consist of the number (or fraction) of the tuples having each of the values of attribute A
- If $V(R, A)$ is too large, only the most frequent values of attribute A may be recorded individually, while other values are counted in groups

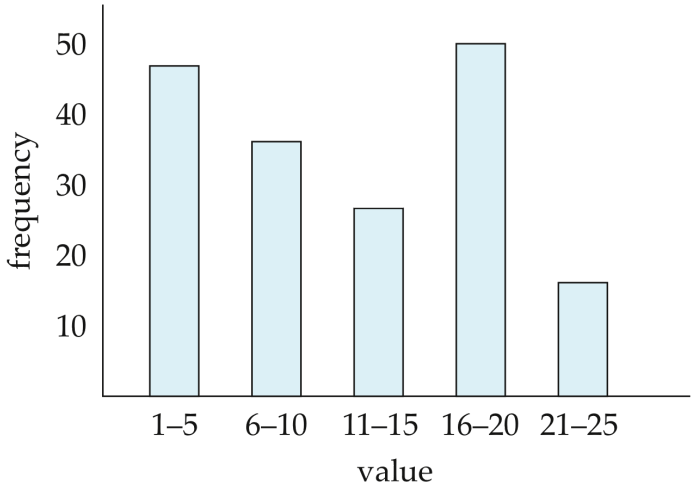


Histograms (Cont'd)

Types of Histograms

- Equal-width histograms
- Equal-height histograms
- Most-frequent-values histograms

The sizes of joins can be estimated more accurately using histograms



Estimating the Size of a Natural Join using Histograms

Example (Cost Estimation using Histograms)

Given the histograms on $R.b$ and $S.b$, estimate $T(R(a,b) \bowtie S(b,c))$

Range	$R.b$	$S.b$
0–9	40	0
10–19	60	0
20–29	80	0
30–39	50	0
40–49	10	5
50–59	5	20
60–69	0	50
70–79	0	100
80–89	0	60
90–99	0	10
Total	245	245

- (Using histograms) $T(R \bowtie S) = 10 \times 5/10 + 5 \times 20/10 = 15$
- (Without histograms) $T(R \bowtie S) = 245 \times 245/100 = 600$

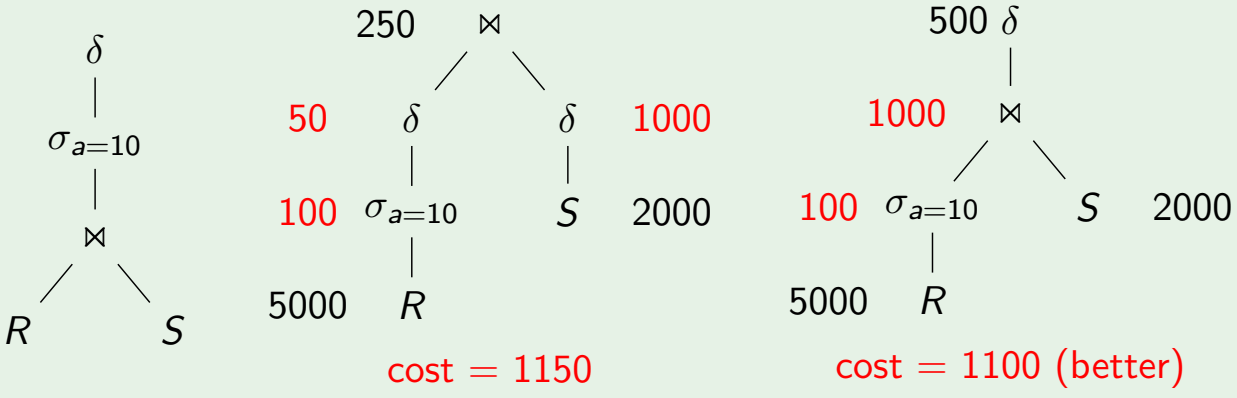
Heuristics for Improving Logical Query Plans

1. Estimate the costs before and after a transformation
2. Apply the transformation if it appears to reduce cost and avoid the transformation otherwise

The cost of a logical plan = the sum of the sizes of intermediate results

Example

- $R(a,b)$: $T(R) = 5000, V(R,a) = 50, V(R,b) = 100$
- $S(b,c)$: $T(S) = 2000, V(S,b) = 200, V(S,c) = 100$



Exercises

Given the statistics of the relations, estimate the sizes of the following expressions

$W(a,b)$	$X(b,c)$	$Y(c,d)$	$Z(d,e)$
$T(W)=100$	$T(X)=200$	$T(Y)=300$	$T(Z)=400$
$V(W,a)=20$	$V(X,b)=50$	$V(Y,c)=50$	$V(Z,d)=40$
$V(W,b)=60$	$V(X,c)=100$	$V(Y,d)=50$	$V(Z,e)=100$

1. $W \bowtie X \bowtie Y \bowtie Z$
2. $\sigma_{a=10}(W)$
3. $\sigma_{c=20}(Y)$
4. $\sigma_{c=20}(Y) \bowtie Z$
5. $W \times Y$
6. $\sigma_{d>10}(Z)$
7. $\sigma_{a=1 \wedge b=2}(W)$
8. $\sigma_{a=1 \wedge b>2}(W)$
9. $\sigma_{a<1 \wedge a>2}(W)$
10. $X \bowtie_{X.c<Y.c} Y$

Exercises

Estimate the size of the join $R(a,b) \bowtie S(b,c)$ using histograms for $R.b$ and $S.b$. Assume $V(R,b) = V(S,b) = 20$.

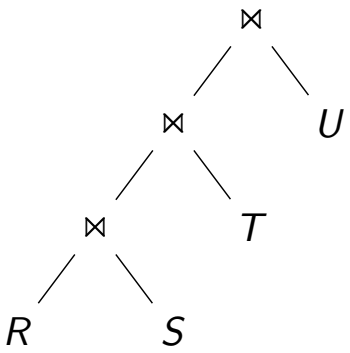
	0	1	2	3	4	others
$R.b$	5	6	4	5	0	32
$S.b$	10	8	5	0	7	48

9.2 Improving Logical Query Plans
9.2.3 Choosing an Order for Joins

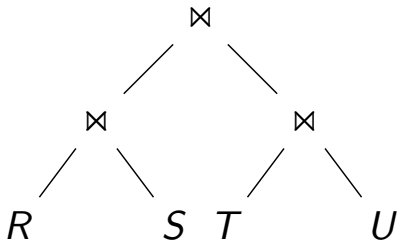
Join Trees (连接树)

A join tree depicts a plan for joining a set of relations

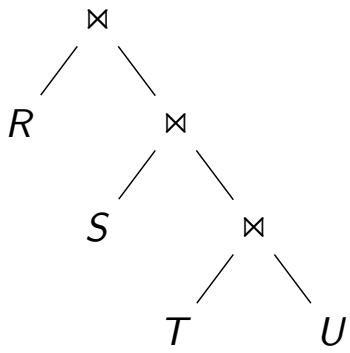
- Left-deep trees: all relations except one are on the right branches
- Right-deep trees: all relations except one are on the left branches
- Bushy trees: other trees except left-deep trees and right-deep trees



Left deep trees



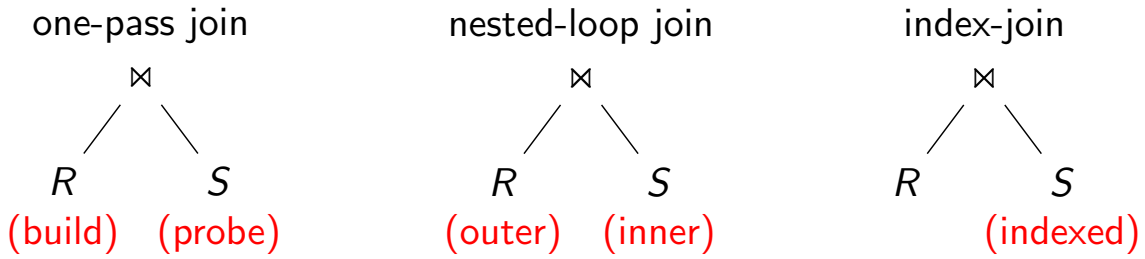
Bushy trees



Right deep trees

Roles of Arguments of Join Operations

- In a join tree, the arguments of a join operation have different roles
- One-pass joins
  - The left argument is the build relation
  - The right argument is the probe relation
- Nested-loop joins
  - The left argument is the outer relation
  - The right argument is the inner relation
- Index-based joins
  - The right argument is the indexed relation



Left-Deep Join Trees

Many DBMSs only consider left-deep trees in query optimization

Advantage 1: The number of possible left-deep trees with a given number of leaves is much smaller than the number of all trees

$T(n)$ = the number of all tree shapes with n leaves

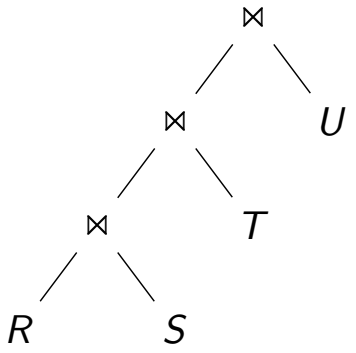
$$T(1) = 1,$$
$$T(n) = \sum_{i=1}^{n-1} T(i)T(n-i) \qquad \text{if } n > 1$$

- The number of all trees with n leaves is thus $n!\,T(n)$
- The number of all left-deep trees with n leaves is $n!$

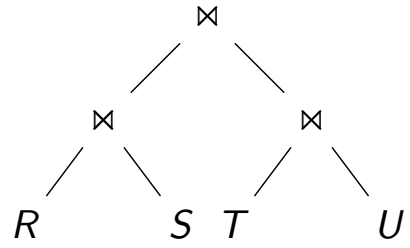
Left-Deep Join Trees (Cont'd)

Advantage 2: Query plans based on left-deep trees plus the selected join algorithms will tend to be more efficient than the same join algorithms used with non-left-deep trees

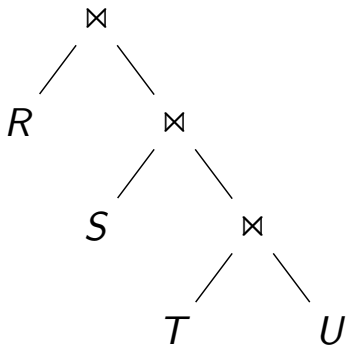
- If we use one-pass joins, the amount of memory needed at any time tends to be smaller than if we used a right-deep tree or a bushy tree for the same relations



At any time, only one of R, $R \bowtie S$, and $R \bowtie S \bowtie T$ will be held in the buffers



$R \bowtie S$ and T must be held in the buffers at the same time

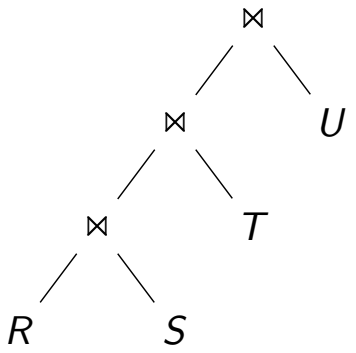


All of R, S, and T must be held in the buffers at the same time

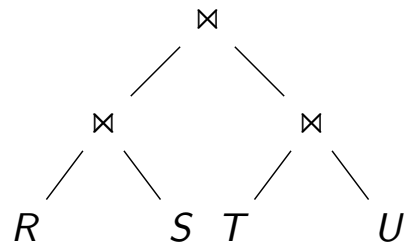
Left-Deep Join Trees (Cont'd)

Advantage 2: Query plans based on left-deep trees plus the selected algorithms will tend to be more efficient than the same algorithms used with non-left-deep trees

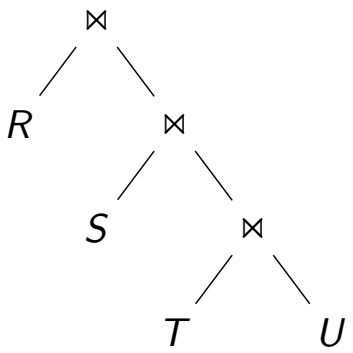
- If we use nested-loop joins, we avoid constructing any intermediate relation more than once



Both $R \bowtie S$ and $R \bowtie S \bowtie T$ are constructed only once



$R \bowtie S$ is constructed only once, but $T \bowtie U$ needs to be constructed more than once (if not materialized)



$S \bowtie T \bowtie U$ needs to be constructed more than once, so does $T \bowtie U$

Dynamic Programming to Select a Join Order and Grouping

Example (Optimization of Join Orders)

Choose the best join order for $R(a,b) \bowtie S(b,c) \bowtie T(c,d) \bowtie U(d,a)$

$R(a,b)$	$S(b,c)$	$T(c,d)$	$U(d,a)$
$T(R) = 1000$	$T(S) = 1000$	$T(T) = 1000$	$T(U) = 1000$
$V(R,a) = 100$			$V(U,a) = 50$
$V(R,b) = 200$	$V(S,b) = 100$		
	$V(S,c) = 500$	$V(T,c) = 20$	
		$V(T,d) = 50$	$V(U,d) = 1000$

Best orders of joins on 2 relations

	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$
Est. size	5,000	1M	10,000	2,000	1M	1,000
Est. cost	0	0	0	0	0	0
Best plan	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$

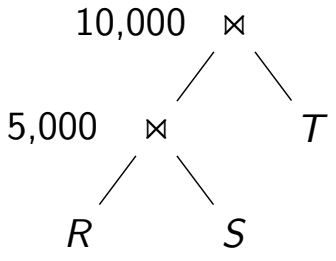
Best orders of joins on 2 relations

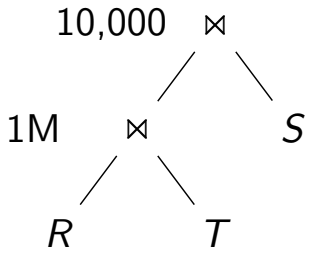
	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$
Est. size	5,000	1M	10,000	2,000	1M	1,000
Est. cost	0	0	0	0	0	0
Best plan	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$

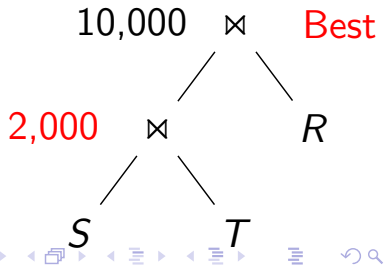
Best orders of joins on 3 relations

	$R \bowtie S \bowtie T$	$R \bowtie S \bowtie U$	$R \bowtie T \bowtie U$	$S \bowtie T \bowtie U$
Est. size	10,000	50,000	10,000	2,000
Est. cost	2,000	5,000	1,000	1,000
Best plan	$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(T \bowtie U) \bowtie S$

For example, the best plan for $R \bowtie S \bowtie T$ is obtained as follows



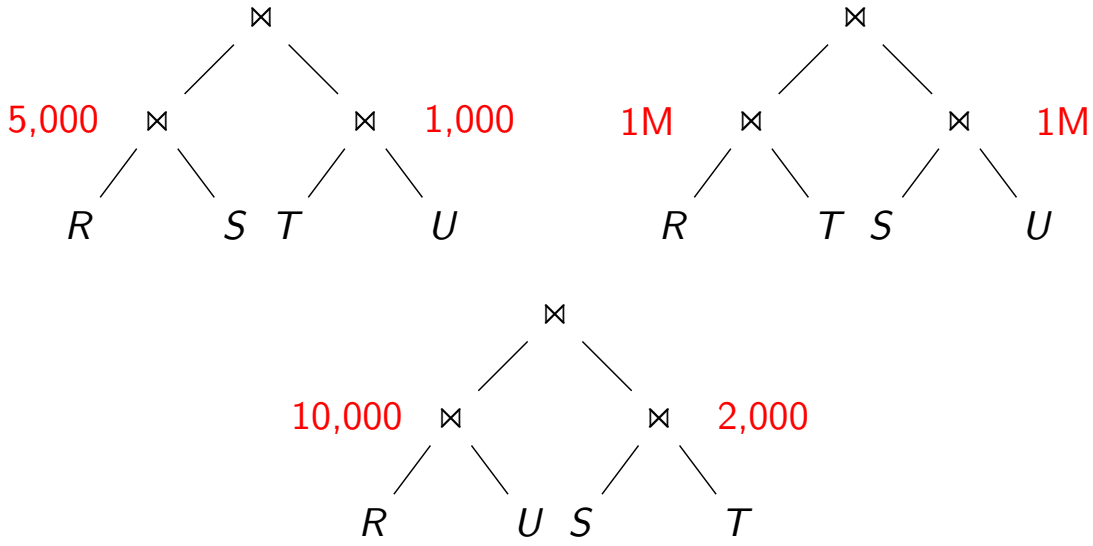




The best plan for $R \bowtie S \bowtie T \bowtie U$ is obtained as follows (phase 1)

Best orders of joins on 2 relations

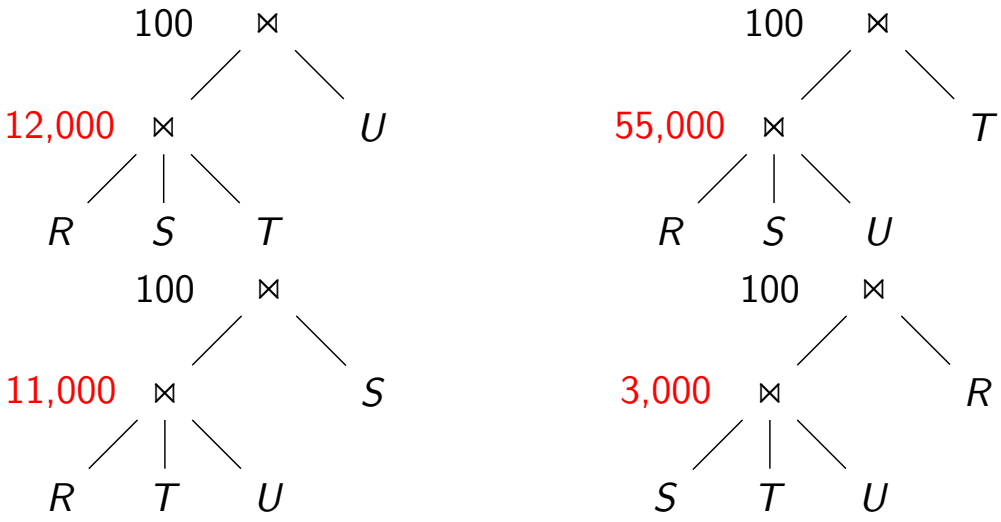
	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$
Est. size	5,000	1M	10,000	2,000	1M	1,000
Est. cost	0	0	0	0	0	0
Best plan	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$



The best plan for $R \bowtie S \bowtie T \bowtie U$ is obtained as follows (phase 2)

Best orders of joins on 3 relations

	$R \bowtie S \bowtie T$	$R \bowtie S \bowtie U$	$R \bowtie T \bowtie U$	$S \bowtie T \bowtie U$
Est. size	10,000	50,000	10,000	2,000
Est. cost	2,000	5,000	1,000	1,000
Best plan	$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(T \bowtie U) \bowtie S$



Excercises

1. If we use one-pass joins, the amount of memory needed at any time tends to be smaller than if we used a right-deep tree or a bushy tree for the same relations. Please explain using an example.
2. If we use nested-loop joins, we avoid constructing any intermediate relation more than once. Please explain using an example.
3. Design a dynamic programming algorithm to find the best left-deep order of joins

9.3 Improving Physical Query Plans

Improving Physical Query Plans

- Select the physical operators used to implement logical operators (Section 9.3.1)
- Select the method of passing arguments from one physical operator to the next (Section 9.3.2)

9.3 Improving Physical Query Plans
9.3.1 Selection of Algorithms for Operators

Choosing a Selection Algorithm for $\sigma_\theta(R)$

- Index Scan Only: If θ is of the form $A = a$, and there is an index on attribute A, we use the index-based selection.
- Index Scan + Intersection: If θ is of the form $A = a \wedge B = b$, and there is an index on each of attributes A and B, we intersect the pointers to the tuples satisfying $A = a$ and $B = b$ retrieved from the indexes on A and B, respectively, and read these tuples from disks.
- Index Scan + Filtering: If θ is of the form $A = a \wedge \theta'$, and there is an index only on attribute A, we first retrieve all tuples that satisfy condition $A = a$ using the index-based selection algorithm and then filter each selected tuple satisfying condition θ'.
- Table Scan: Other cases.

Choosing a Selection Algorithm for $\sigma_\theta(R)$

Example

- $T(R) = 5000, B(R) = 200, V(R, x) = 100, V(R, y) = 500$
- Query: $\sigma_{x=1 \wedge y=2 \wedge z<3}(R)$
- Table Scan: $B(R) = 200$ I/O's
- Index Scan on x + Filtering: $T(R)/V(R, x) = 50$ I/O's
- Index Scan on y + Filtering: $T(R)/V(R, y) = 10$ I/O's (The best)
- Index Scan on z + Filtering: $B(R)/3 = 67$ I/O's

Choosing a Join Algorithm for $R \bowtie S$

- One-Pass Join: Either R or S fits in main memory
- Nested-Loop Join: None of R and S fit in main memory
- Sort-Merge Join: One or both of R and S are already sorted on their join attribute(s)
- Sort-Merge Join: There are two or more joins on the same attribute such as $R(a, b) \bowtie S(a, c) \bowtie T(a, d)$
- Index-Join: R is expected to be small, and there is an index on the join attribute(s) for S
- Hash-Join: There is no opportunity to use already-sorted relations or indexes, and a multipass join is needed

9.3 Improving Physical Query Plans
9.3.2 Materialization vs. Pipelining

Materialization (物化法)

- Evaluate one operation at a time, in an appropriate order
- The result of each evaluation is materialized in a temporary relation for subsequent use

Disadvantages

- Temporary relations must be written to disk and later read back (unless they are small), which increases the cost of query evaluation
- There may be a long delay before a user sees any query results

Example (Materialization)

$\Pi_{ID,salary}$ ⇢ (output relation)

|

$\sigma_{salary<75000}$; use index ⇢ (temporary relation)

|

instructor ⇢ (input relation)

Piplining (流水线法)

- Goal: reduce the number of temporary relations that are produced
- Idea: combine several relational operations into a pipeline of operations, in which the results of one operation are passed along to the next operation in the pipeline
- Advantages
  - It eliminates the cost of reading and writing temporary relations
  - It can start generating query results quickly

Example (Pipelining)

$\Pi_{ID,salary}$ ⇢ (output relation)

| pipelined

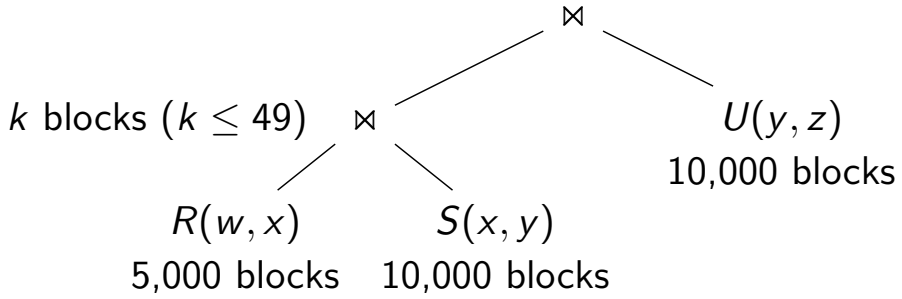
$\sigma_{salary<75000}$; use index (~~temporary relation~~)

|

instructor ⇢ (input relation)

Piplining (Cont'd)

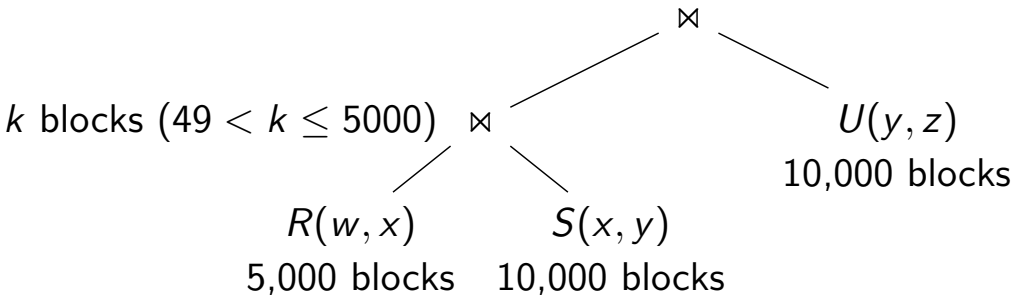
There are $M = 101$ blocks available for use in the buffer pool



1. Executing $R \bowtie S$ using hash-join
   - I/O cost $= 3B(R) + 3B(S) = 45000$
   - Memory cost of phase 1 $= 101$ blocks (100 for sublist, and 1 for input)
   - Memory cost of phase 2 $= 51$ blocks (50 for storing each sublist of R, and 1 for reading S)
2. Piplining $R \bowtie S$ to $(R \bowtie S) \bowtie U$
   - Storing $R \bowtie S$ needs $k \leq 49$ blocks
3. Executing $(R \bowtie S) \bowtie U$ using one-pass join
   - I/O cost $= B(U) = 10000$ ($R \bowtie S$ is already in main memory)
   - Memory cost $= 50$ blocks (49 for storing $R \bowtie S$ and 1 for reading U)

Piplining (Cont'd)

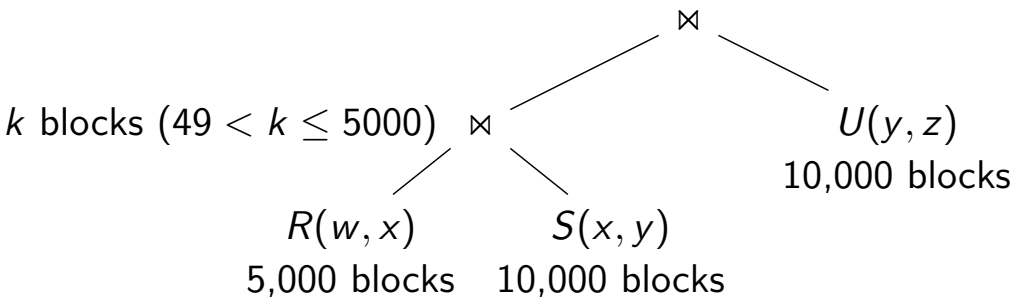
There are $M = 101$ blocks available for use in the buffer pool



1. Executing $R \bowtie S$ using hash-join
   - I/O cost $= 3B(R) + 3B(S) = 45000$
   - Memory cost of phase 1 $= 101$ blocks
   - Memory cost of phase 2 $= 51$ blocks
2. Piplining $R \bowtie S$ to $(R \bowtie S) \bowtie U$
   - Hashing $R \bowtie S$ into 50 buckets
   - I/O cost $= k$ (writing 50 buckets to disk)
3. Executing $(R \bowtie S) \bowtie U$ using hash join
   - I/O cost $= B(R \bowtie S) + 3B(U) = k + 30000$
   - Memory cost $= \lceil k/50 \rceil + 1$ blocks ($\lceil k/50 \rceil$ for $R \bowtie S$ and 1 for U)

Excercises

There are $M = 101$ blocks available for use in the buffer pool



What is the cost of the following query execution plan?

1. Executing $R \bowtie S$ using hash-join
2. Piplining $R \bowtie S$ to $(R \bowtie S) \bowtie U$
3. Executing $(R \bowtie S) \bowtie U$ using nested-loop join

Summary

1. 9.1 Overview of Query Optimization
2. 9.2 Improving Logical Query Plans
   - 9.2.1 Transformations of Relational-Algebra Expressions
   - 9.2.2 Estimating the Cost of Operations
   - 9.2.3 Choosing an Order for Joins
3. 9.3 Improving Physical Query Plans
   - 9.3.1 Selection of Algorithms for Operators
   - 9.3.2 Materialization vs. Pipelining