第10章：查询优化

Query Optimization

邹兆年

哈尔滨工业大学
计算机科学与技术学院
海量数据计算研究中心
电子邮件：znzou@hit.edu.cn

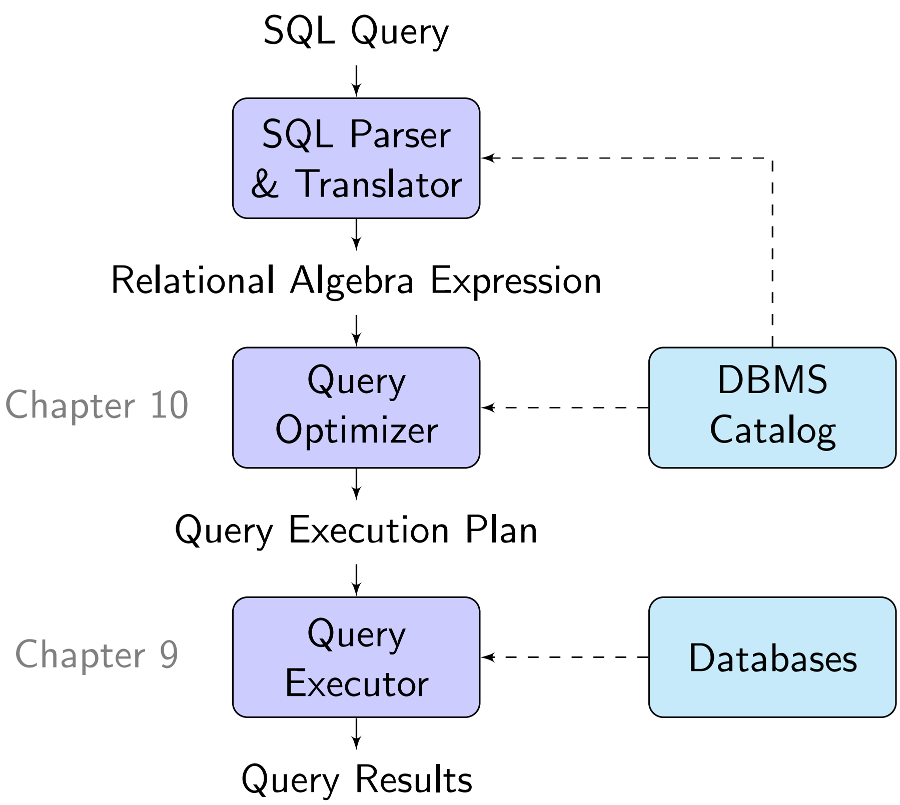
2021年春

教学内容[1]

[1] 课件更新于2021年4月19日

Overview

查询处理(Query Processing)



查询优化(Query Optimization)

查询优化(query optimization): 将一个关系代数表达式转换成一个可以快速执行的物理查询计划(physical query plan)的过程

- 查询优化是一个NP难问题

查询优化器(query optimizer): DBMS中负责进行查询优化的组件

- 查询优化器是DBMS中最难设计的组件之一

查询优化的两个阶段

- 逻辑查询优化(logical query optimization)
- 物理查询优化(physical query optimization)

逻辑查询优化(Logical Query Optimization)

将一个逻辑查询计划(logical query plan)转换成另一个具有更低估计代价的逻辑查询计划

Example (逻辑查询优化)

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

初始逻辑查询计划		更好的逻辑查询计划
$\sigma_{salary<75000}$		$\Pi_{ID,salary}$
|	$\Longrightarrow$	|
$\Pi_{ID,salary}$		$\sigma_{salary<75000}$
|		|
instructor		instructor

物理查询优化(Physical Query Optimization)

在选定的逻辑查询计划的基础上，生成一个优化的物理查询计划(physical query plan)，使DBMS按照该物理查询计划执行可以快速得到查询结果

Example (物理查询优化)

逻辑查询计划		物理查询计划
$\Pi_{ID,salary}$		$\Pi_{ID,salary}$
|		|
$\sigma_{salary<75000}$	$\Longrightarrow$	$\sigma_{salary<75000}$; use index
|		|
instructor		instructor

查询优化技术的发展

20世纪70年代，IBM System R最早实现了DBMS的查询优化器

- System R查询优化器的很多概念和技术仍被现在的DBMS使用

如今，人们希望借助机器学习(machine learning)技术来提高查询优化器的准确率和效率

- AI4DB是否比NLP和CV更有挑战性?

Improving Logical Query Plans

逻辑查询优化(Logical Query Optimization)

将一个逻辑查询计划转换成另一个具有更低估计代价的逻辑查询计划

方法1：启发式方法

- 根据规则，将一个逻辑查询计划等价变换为更高效的逻辑查询计划
- 该方法只需知道数据库模式，而无需了解数据库实例的数据分布

方法2：基于代价的方法

- 使用代价估计模型来估计逻辑查询计划的代价(cost)
- 估计多个逻辑查询计划的代价，选择代价最低的逻辑查询计划

逻辑查询优化的关键技术

- 技术1：关系代数表达式的等价变换
- 技术2：逻辑查询计划的代价估计(cost estimation)
- 技术3：连接顺序(join order)优化

Improving Logical Query Plans
Transformations of Relational Algebra Expressions

等价关系代数表达式

如果两个关系代数表达式在任意数据库实例上的结果都相同，则这两个关系代数表达式等价(equivalent)

Example (等价关系代数表达式)

初始关系代数表达式树 ≡ 等价关系代数表达式树

$$\begin{array}{c} \sigma_{salary<75000} \\ | \\ \Pi_{ID,salary} \\ | \\ \text{instructor} \end{array} \quad \equiv \quad \begin{array}{c} \Pi_{ID,salary} \\ | \\ \sigma_{salary<75000} \\ | \\ \text{instructor} \end{array}$$

关系代数表达式的等价变换/查询改写(Query Rewriting)

将一个关系代数表达式转换成另一个(通常具有更高效的物理查询计划的)等价关系代数表达式

- 只需知道数据库模式
- 无需查询计划代价模型

Example (关系代数表达式的等价变换)

初始关系代数表达式树 ⟹ 等价关系代数表达式树

$$\begin{array}{c} \sigma_{salary<75000} \\ | \\ \Pi_{ID,salary} \\ | \\ \text{instructor} \end{array} \quad \Longrightarrow \quad \begin{array}{c} \Pi_{ID,salary} \\ | \\ \sigma_{salary<75000} \\ | \\ \text{instructor} \end{array}$$

关系代数表达式的等价变换规则

既满足交换律又满足结合律的关系代数操作

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

θ连接满足交换律，但不满足结合律(见作业1)

- $(R(a,b) \bowtie_{R.b>S.b} S(b,c)) \bowtie_{a<d} T(c,d) \neq R(a,b) \bowtie_{R.b>S.b} (S(b,c) \bowtie_{a<d} T(c,d))$

有关选择的等价变换规则

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

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

有关选择的等价变换规则(续)

Pushing a selection through a product

- $\sigma_\theta(R \times S) = R \bowtie_\theta S$
- $\sigma_\theta(R \times S) = \sigma_\theta(R) \times S$
  如果R包含θ中使用的全部属性，而S没有

Pushing a selection through a theta-join

- $\sigma_{\theta_1}(R \bowtie_{\theta_2} S) = R \bowtie_{\theta_1 \wedge \theta_2} S$
- $\sigma_{\theta_1}(R \bowtie_{\theta_2} S) = \sigma_{\theta_1}(R) \bowtie_{\theta_2} S$
  如果R包含θ_1中使用的全部属性，而S没有

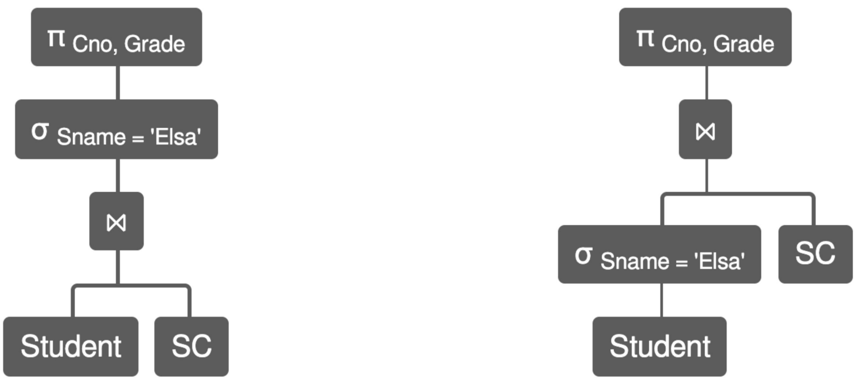
Pushing a selection through a natural join

- $\sigma_\theta(R \bowtie S) = \sigma_\theta(R) \bowtie S$
  如果R包含θ中使用的全部属性，而S没有
- $\sigma_\theta(R \bowtie S) = \sigma_\theta(R) \bowtie S = R \bowtie \sigma_\theta(S) = \sigma_\theta(R) \bowtie \sigma_\theta(S)$
  如果R和S均包含θ中使用的全部属性

选择下推(Selection Pushdown)

将关系代数表达式树中的选择操作向下推(push down)通常可以提高查询的执行效率

- 选择下推可以尽早过滤掉与结果无关的元组

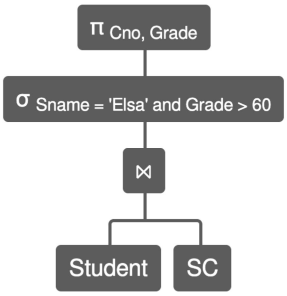


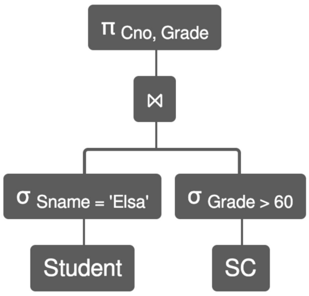
$\pi_{\text{Cno, Grade}}(\sigma_{\text{Sname = 'Elsa'}}(\text{Student} \bowtie \text{SC}))$ $\pi_{\text{Cno, Grade}}(\sigma_{\text{Sname = 'Elsa'}}(\text{Student}) \bowtie \text{SC})$

选择下推(续)

将复杂的选择条件进行分解，然后再进行选择下推

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



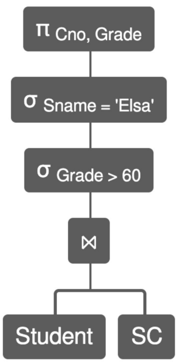


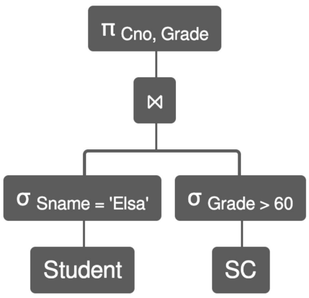
$\pi_{Cno,\ Grade}(\sigma_{Sname\ =\ 'Elsa'\ and\ Grade\ >\ 60}(Student \bowtie SC))$ $\pi_{Cno,\ Grade}(\sigma_{Sname\ =\ 'Elsa'}(Student) \bowtie \sigma_{Grade\ >\ 60}(SC))$

选择下推(续)

选择度高的选择操作优先做

- 选择度(selectivity)：属性不同值的个数与元组数的比值
- 尽早过滤掉更多与结果无关的元组





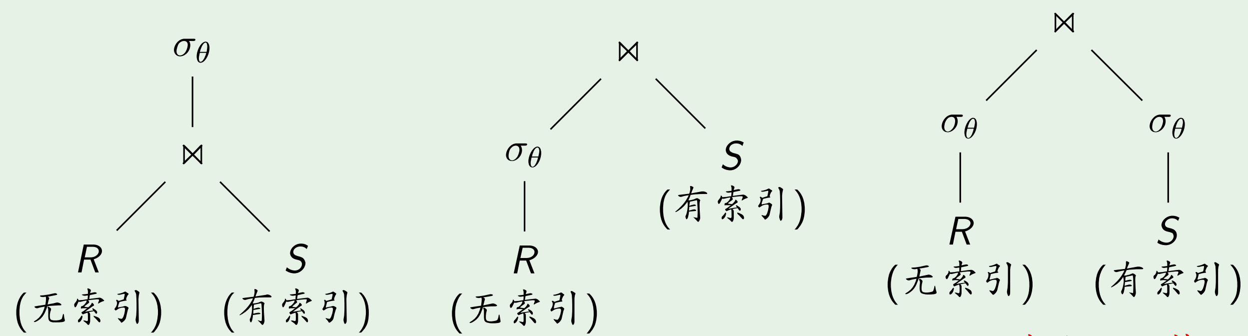
$\pi_{Cno,\ Grade}(\sigma_{Sname\ =\ 'Elsa'}(\sigma_{Grade\ >\ 60}(Student \bowtie SC)))$ $\pi_{Cno,\ Grade}(\sigma_{Sname\ =\ 'Elsa'}(Student) \bowtie \sigma_{Grade\ >\ 60}(SC))$

选择下推(续)

当选择操作可以向关系代数表达式树的多个分支下推时，需要考虑向哪个分支下推更合适

- 索引会影响选择下推的方案
- 选择操作的结果上没有索引

Example (索引对选择下推的影响)



初始关系代数表达式树　　好的选择下推　　不好的选择下推 ($\sigma_\theta(S)$上无索引)

选择条件的改写

改写低效的选择条件

- 改写前：`X = Y AND X = 3`
  改写后：`X = 3 AND Y = 3`

去掉不必要的选择条件

- 改写前：`SELECT * FROM R WHERE 1 = 1;`
  改写后：`SELECT * FROM R;`

检查无法满足的选择条件

- `SELECT * FROM R WHERE 1 = 0;`

选择条件的改写(续)

合并选择条件

- 改写前:

```
SELECT * FROM R
WHERE A BETWEEN 1 AND 100
OR A BETWEEN 50 AND 150;
```

改写后:

```
SELECT * FROM R
WHERE A BETWEEN 1 AND 150;
```


有关投影的等价变换规则

Pushing a projection through a projection (the absorbing law)

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

Pushing a projection through a selection

- $\Pi_L(\sigma_\theta(R)) = \Pi_L(\sigma_\theta(\Pi_M(R)))$
  M中既包含L中的属性，又包含θ中使用的属性

Pushing a projection through a union

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

有关投影的等价变换规则(续)

Pushing a projection through a product

- $\Pi_L(R \times S) = \Pi_L(\Pi_M(R) \times \Pi_N(S))$
  M中包含既在R中又在L中的属性
  N中包含既在S中又在L中的属性

Pushing a projection through a natural join

- $\Pi_L(R \bowtie S) = \Pi_L(\Pi_M(R) \bowtie \Pi_N(S))$
  M中包含既在R中又在L中的属性，以及R中连接属性
  N中包含既在S中又在L中的属性，以及S中连接属性

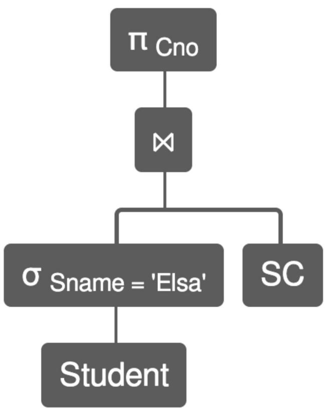
Pushing a projection through a theta-join

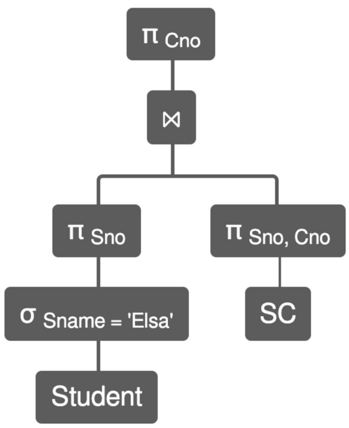
- $\Pi_L(R \bowtie_\theta S) = \Pi_L(\Pi_M(R) \bowtie_\theta \Pi_N(S))$
  M中包含既在R中又在L中的属性，以及R中连接属性
  N中包含既在S中又在L中的属性，以及S中连接属性

投影下推(Projection Pushdown)

将关系代数表达式树中的投影操作向下推通常可以提高查询执行效率

- 投影下推可以降低元组的大小





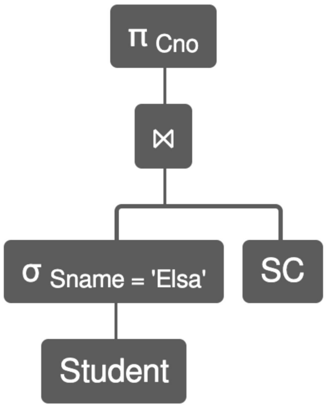
$\pi_{Cno}(\sigma_{Sname = 'Elsa'}(Student) \bowtie SC)$ $\pi_{Cno}(\pi_{Sno}(\sigma_{Sname = 'Elsa'}(Student)) \bowtie \pi_{Sno, Cno}(SC))$

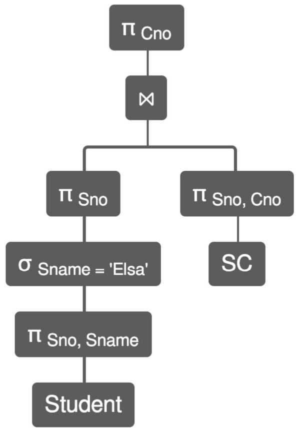
投影下推(续)

有些情况下，投影操作下推会浪费查询优化的机会

- 投影操作的结果上没有索引

- 假设关系Student上建有属性Sname上的二级索引
- $\Pi_{Sno,Sname}(Student)$使后续的选择和连接操作均无法利用该索引





$\pi_{Cno}(\sigma_{Sname='Elsa'}(Student) \bowtie SC)$ $\pi_{Cno}(\pi_{Sno}(\sigma_{Sname='Elsa'}(\pi_{Sno,Sname}(Student))) \bowtie \pi_{Sno,Cno}(SC))$

查询改写

去除不必要的投影

- 改写前:

```
SELECT * FROM SC WHERE EXISTS (
  SELECT Sno FROM Student
    WHERE Sname = 'Elsa'
    AND Student.Sno = SC.Sno);
```

改写后:

```
SELECT * FROM SC WHERE EXISTS (
  SELECT * FROM Student
    WHERE Sname = 'Elsa'
    AND Student.Sno = SC.Sno);
```

查询改写(续)

去除不必要的连接

- 改写前:

```
SELECT Student.*
  FROM Student LEFT JOIN SC;
```

改写后:

```
SELECT * FROM Student;
```


Improving Logical Query Plans
Estimation of Query Plan Cost

查询计划的代价(Query Plan Cost)

查询计划的代价(query plan cost)

- 反映查询计划执行时间长短的数
- 查询计划的代价越低，其执行时间越短
- 查询计划的代价与实际执行时间没有直接的函数关系
- 比较不同查询的查询计划的代价没有任何意义

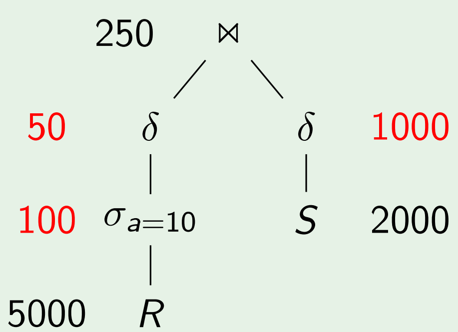
查询计划代价的度量指标

查询计划的代价用查询计划执行过程中产生的中间结果的元组数来度量

- 查询计划的执行时间$\xleftarrow{estimate}$执行查询计划时产生的I/O次数
- I/O次数$\xleftarrow{estimate}$查询计划执行过程中产生的中间结果的大小
- 中间结果的大小$\xleftarrow{estimate}$中间结果的基数(cardinality)，即元组数

Example (查询计划的代价)

下面的查询计划的代价$= 50 + 100 + 1000 = 1150$



查询计划的代价估计

查询计划的代价估计归结为操作结果的基数估计(cardinality estimation)

- 查询优化器根据操作的输入关系的元组数来估计操作结果的元组数

操作结果基数估计方法的特性

- 准确
- 易计算
- 逻辑一致性(logically consistent)

逻辑一致性

- 单调性：一个操作的输入越大，操作结果基数的估计值越大
- 顺序无关：在多个关系上执行同一种满足交换律和结合律的操作时(如$\bowtie, \times, \cup, \cap$)，最终结果基数的估计值与操作的执行顺序无关

操作结果基数估计所需的统计信息

系统目录(system catalog)中记录着与操作结果基数估计有关的统计信息

- $T(R)$：关系R的元组数
- $V(R, A)$：关系R的属性集A的不同值的个数

基本假设

- 关系中每个属性的取值均服从均匀分布
- 关系的所有属性相互独立

操作结果基数估所需的统计信息(续)

DBMS会自动收集统计信息

在DBMS中亦可手动收集统计信息

- Postgres/SQLite: `ANALYZE`
- Oracle/MySQL: `ANALYZE TABLE`
- SQL Server: `UPDATE STATISTICS`
- DB2: `RUNSTATS`

笛卡尔积操作的基数估计

- $T(R \times S) = T(R)T(S)$

投影操作的基数估计

不含去重的投影操作

- $T(\Pi_L(R)) = T(R)$

含去重的投影操作

- $T(\Pi_L(R)) = V(R, L)$

选择操作的基数估计

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

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

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

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

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

选择操作的基数估计(续)

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

- $S = \sigma_{\theta_1 \wedge \theta_2}(R) = \sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_2}(\sigma_{\theta_1}(R))$
- $T(S) = T(R) f_1 f_2$ (假设θ_1和θ_2独立)

$$f_i = \begin{cases} 1/V(R,A) & \text{if } \theta_i \text{ is of the form } A = c, \\ 1 - 1/V(R,A) & \text{if } \theta_i \text{ is of the form } A \neq c, \\ 1/3 & \text{if } \theta_i \text{ is of the form } A < c \text{ or } A > c, \end{cases}$$

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

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

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

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

二路自然连接(2-Way Natural Join)的基数估计

考虑两个关系R和S的自然连接$R \bowtie S$

基本假设

- 连接属性值集合包含假设(Containment of Value Sets)
  对于连接属性K，如果$V(R,K) \leq V(S,K)$，则R的K属性值集合是S的K属性值集合的子集
- 非连接属性值集合保留假设(Preservation of Value Sets)
  对于R中任意非连接属性A，有$V(R \bowtie S, A) = V(R,A)$

二路自然连接的基数估计(续)

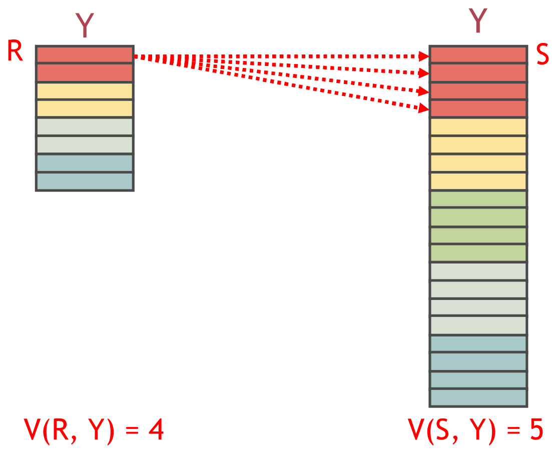
情况1: R和S只有一个连接属性Y

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

证明

不失一般性，假设$V(R,Y) \le V(S,Y)$

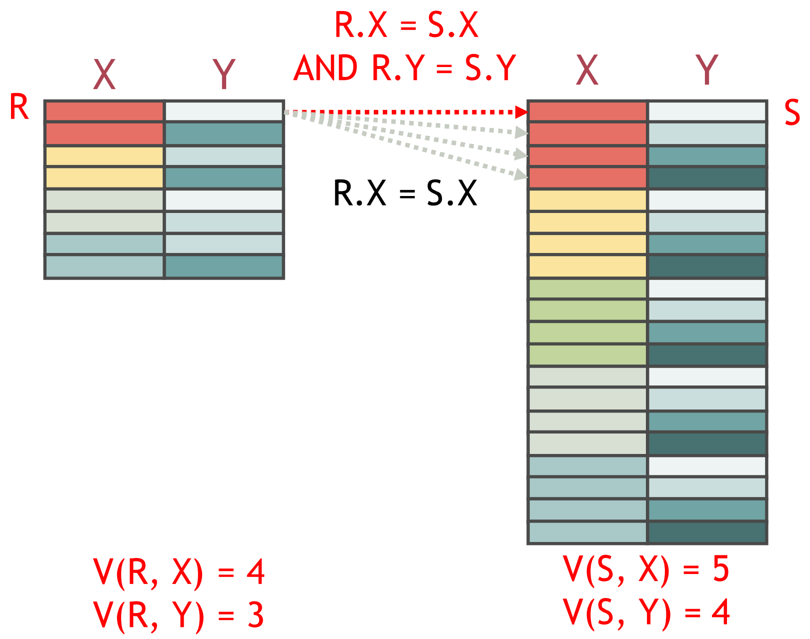
- 根据属性值包含假设，R的Y属性值集合是S的Y属性值集合的子集，因此R的每个元组都可以和S的一些元组进行连接
- 根据属性值均匀分布假设，R的每个元组都可以和S中$\frac{T(S)}{V(S,Y)}$个元组进行连接
- 因此，$R \bowtie S$的结果包含$\frac{T(R)T(S)}{V(S,Y)}$个元组



二路自然连接的基数估计(续)

情况2: R和S有2个连接属性X和Y

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



证明

设$U = R \bowtie_{R.X=S.X} S$，则$T(R \bowtie S) = T(\sigma_{R.Y=S.Y}(U))$

- $T(U) = \frac{T(R)T(S)}{\max(V(R,X),V(S,X))}$
- 根据属性值保留假设，有$V(U,R.Y) = V(R,Y)$，$V(U,S.Y) = V(S,Y)$
- 不失一般性，假设$V(R,Y) \leq V(S,Y)$，则$V(U,R.Y) \leq V(U,S.Y)$
- 根据属性值包含假设，U的$R.Y$属性值集合是U的$S.Y$属性值集合的子集
- 根据属性值均匀分布假设，U中满足$R.Y = S.Y$的元组占$1/V(U,S.Y) = 1/V(S,Y)$
- 因此，$T(\sigma_{R.Y=S.Y}(U)) = \frac{T(R)T(S)}{\max(V(R,X),V(S,X)) \cdot V(S,Y)}$

二路自然连接的基数估计(续)

情况3: R和S有2个以上连接属性

证明留作课后习题

习题

Example (二路连接的基数估计)

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

- $T((R \bowtie S) \bowtie U) = ?$
- $T(R \bowtie (S \bowtie U)) = ?$
- $T((R \bowtie U) \bowtie S) = ?$
- 上述二路连接操作的基数估计方法满足逻辑一致性吗?

多路自然连接(Multi-way Natural Join)的基数估计

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

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

设$R_{i_1}, R_{i_2}, \ldots R_{i_n}$是所有包含属性$A$的关系，
且$V(R_{i_1}, A) \le V(R_{i_2}, A) \le \cdots \le V(R_{i_n}, A)$

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

自然连接操作基数估计的逻辑一致性

无论是按不同的顺序进行多次二路连接，还是进行1次多路连接，上述基数估计方法得到的结果相同

Example (连接操作的基数估计)

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

- $T((R \bowtie S) \bowtie U) = ?$
- $T(R \bowtie (S \bowtie U)) = ?$
- $T((U \bowtie R) \bowtie S) = ?$
- $T(R \bowtie S \bowtie U) = ?$

集合操作的基数估计

$T(R \cup S) = \frac{1}{2}(\max(T(R), T(S)) + T(R) + T(S))$

- $\max(T(R), T(S)) \leq T(R \cup S) \leq T(R) + T(S)$

$T(R - S) = T(R) - T(S)/2$

- $T(R) - T(S) \leq T(R - S) \leq T(R)$

$T(R \cap S) = \min(T(R), T(S))/2$

- $0 \leq T(R \cap S) \leq \min(T(R), T(S))$

$T(R \cap S) = T(R \bowtie S)$

- $R \cap S = R \bowtie S$

去重操作的基数估计

$T(\delta(R)) = (T(R) + 1)/2 \approx T(R)/2$

- $1 \leq T(\delta(R)) \leq T(R)$

$T(\delta(R)) = \min((T(R) + 1)/2, V(R, a_1)V(R, a_2) \cdots V(R, a_n))$

- $a_1, a_2, \ldots, a_n$是R的属性
- $1 \leq T(\delta(R)) \leq V(R, a_1)V(R, a_2) \cdots V(R, a_n)$

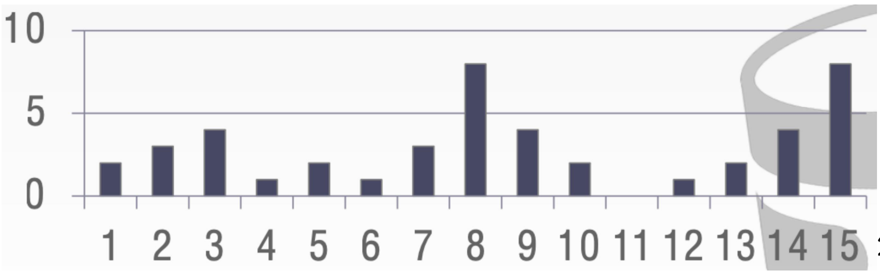
非均匀分布属性的统计信息

上面介绍的基数估计方法全部基于属性值均匀分布假设

- 只需维护$V(R, A)$即可

更准确的基数估计需要对属性值的分布进行更精确的近似

- 方法1：对属性值的出现频率进行计数
- 方法2：维护属性值分布的直方图(histogram)



[2]

[2] 来源：Andy Pevlo, CMU 15-445

属性值出现频率计数

估计一个关系实例R中属性A的每个值v的出现频率

Count-Min算法

- **初始化：**选择值域$\{1, 2, \ldots, n\}$上的m个哈希函数$h_1, h_2, \ldots, h_m$，其中$m, n \ll |R|$。创建一个m行n列的表T，其中每个元素$T[i][j] = 0$
- **离线计数：**对R中每个元组的A属性值v，对$i = 1, 2, \ldots, m$，计算$h_i(v)$并将$T[i][h_i(v)]$加1
- **在线查询：**对给定的A属性值v，v的出现频率估计为

$$\min_{i=\{1,2,\ldots,m\}} T[i][h_i(v)]$$

		1	2	3	4	5
$h_1(v) = 2$	1		+1			
$h_2(v) = 3$	2			+1		
$h_3(v) = 1$	3	+1				
$h_1(v) = 5$	4					+1

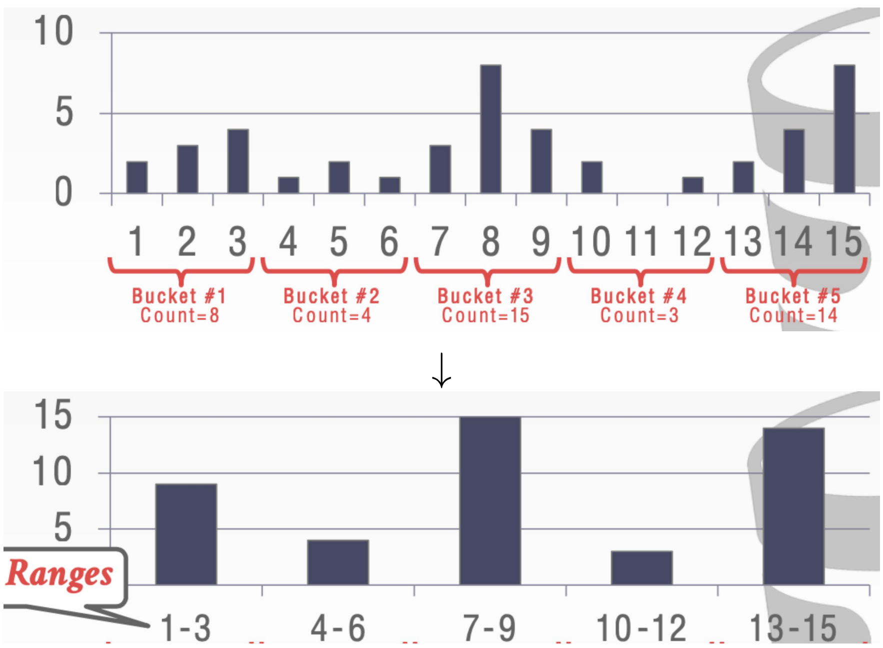
直方图(Histograms)

直方图(histogram)刻画了属性值在不同区间内的出现频率

- 等宽直方图(equal-width histogram)
- 等高直方图(equal-height histogram)

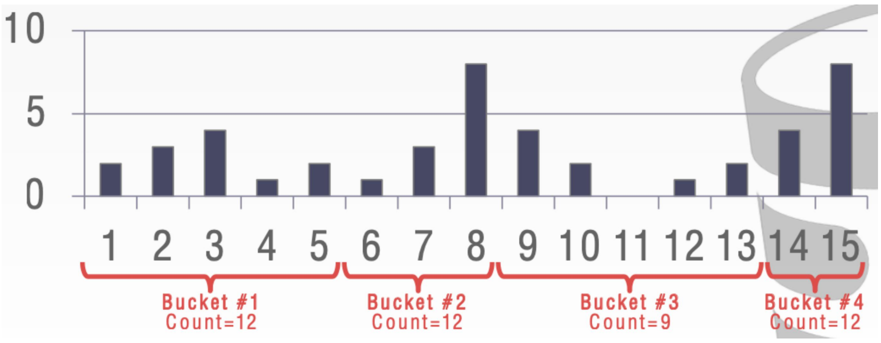
等宽直方图(Equal-Width Histograms)

- 属性值各区间的宽度相同
- 各区间内属性值的出现次数不同

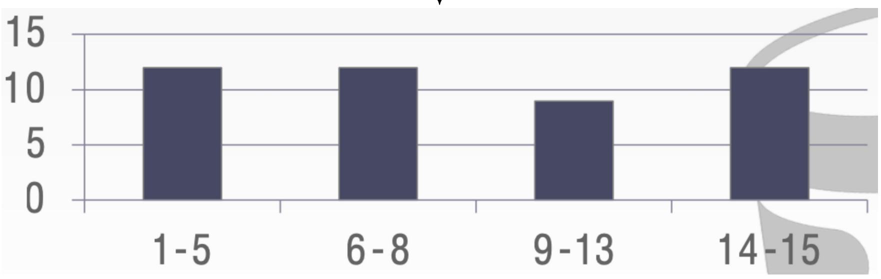


等高直方图(Equal-Height Histograms)

- 属性值各区间的宽度不同
- 各区间内属性值的出现次数基本相同



↓



基于直方图的连接结果基数估计

Example (基于直方图的连接结果基数估计)

已知关系$R(a,b)$和$S(b,c)$的属性$R.b$和$S.b$的直方图如下

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

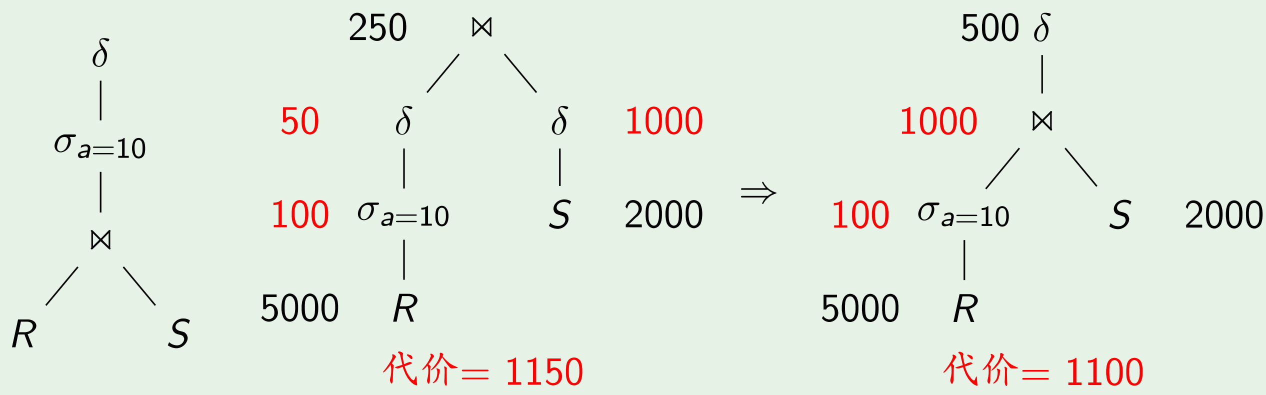
- 使用直方图：$T(R \bowtie S) = 10 \times 5/10 + 5 \times 20/10 = 15$
- 不使用直方图：$T(R \bowtie S) = 245 \times 245/100 = 600$

逻辑查询计划的启发式优化方法

如果某个等价变换能降低查询计划的代价，则对查询计划进行该变换

Example (逻辑查询计划的启发式优化方法)

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



Improving Logical Query Plans
Optimization of Join Orders

连接顺序的优化

优化连接操作的执行顺序对于提高查询执行效率至关重要

- 连接操作的执行代价高
- 在数据库上执行几十个关系的连接很常见，如联机分析处理(Online Analytical Processing, OLAP)任务
- 不同连接顺序的执行代价差异巨大

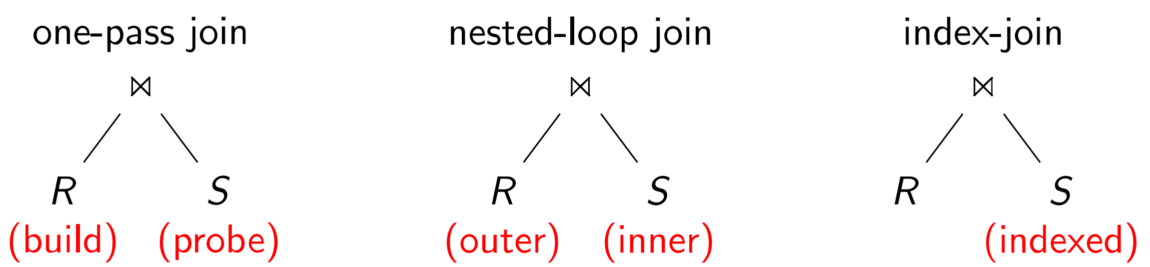
Tips

在PostgreSQL中将配置参数`join_collapse_limit`设置为1可以阻止查询优化器对查询语句中的表连接顺序进行优化(外连接除外)

连接关系的角色

尽管连接操作满足交换律，但在查询计划中连接操作的输入关系具有不同的作用

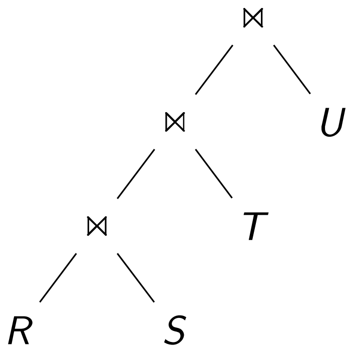
- 在$R \bowtie S$中，R是左关系(left relation)，S是右关系(right relation)
- 如果使用一趟连接(one-pass join)，则左关系是构建关系(build relation)，右关系是探测关系(probe relation)
- 如果使用嵌套循环连接(nested-loop join)，则左关系是外关系(outer relation)，右关系是内关系(inner relation)
- 如果使用索引连接(index-based join)，则右关系是有索引的关系(indexed relation)



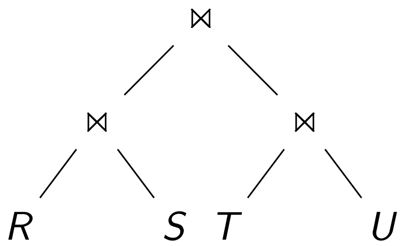
连接树(Join Trees)

一组关系上的连接操作的执行顺序可以用连接树(join tree)来表示

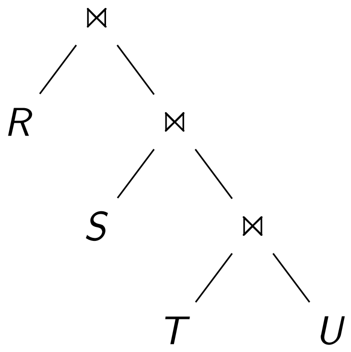
- 左深连接树(left-deep join tree): 只有一个关系是左关系，其他关系都是右关系
- 右深连接树(right-deep join tree): 只有一个关系是右关系，其他关系都是左关系
- 浓密树(bushy tree): 除左深连接树和右深连接树以外的其他连接树



左深连接树



浓密树



右深连接树

左深连接树(Left-Deep Join Trees)

System R的查询优化器只考虑左深连接树

优点1: 在给定关系上，全部左深连接树的数量比全部连接树少得多

- $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$的全部左深连接树的数量为$n!$
- $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$的全部连接树的数量为$n!C_n$，其中$C_n$表示含有$n$个叶子节点的树结构的数量(Catalan数)

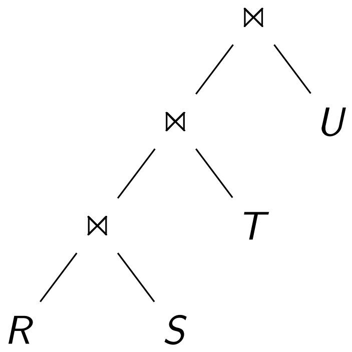
$$C_1 = 1,$$
$$C_n = \sum_{i=1}^{n-1} C_i C_{n-i} \qquad \text{if } n > 1$$

优点2: 基于左深连接树的查询计划通常比基于其他类型连接树的查询计划的执行效率更高

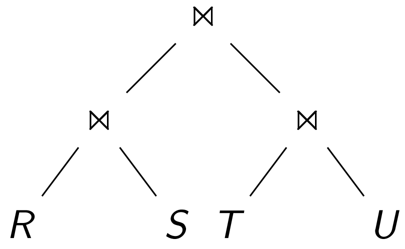
- 可能形成完全通畅的流水线(fully piplined plans)

为什么左深连接树查询计划更高效?

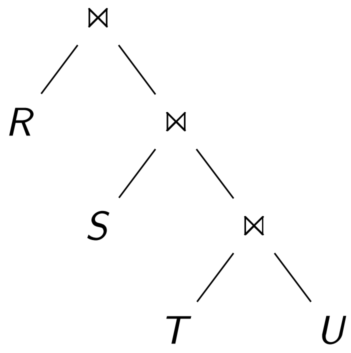
如果下列查询计划全部使用一趟连接(one-pass join)算法，那么在流水线查询执行模型(piplining model)下，左深连接树查询计划在任何时刻对内存的需求都比其他查询计划更低



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



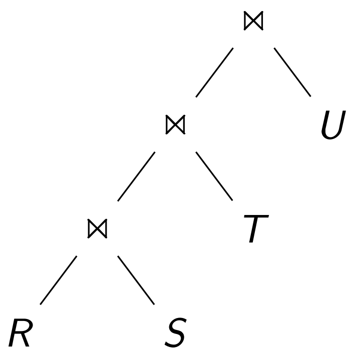
$R \bowtie S$ and T must be held in the buffers at the same time



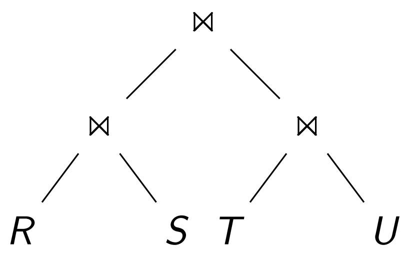
All of R, S, and T must be held in the buffers at the same time

为什么左深连接树查询计划更高效?

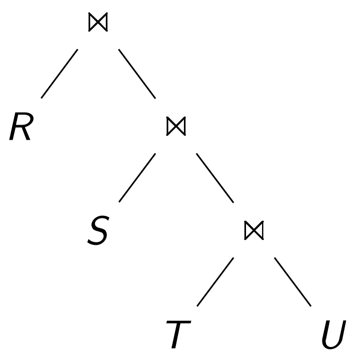
如果下列查询计划全部使用嵌套循环连接(nested-loop join)算法，那么在流水线查询执行模型(piplining model)下，左深连接树查询计划不需要多次构建中间关系



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 $T \bowtie U$

连接顺序优化的动态规划方法

Example (连接顺序优化)

为$R(a,b) \bowtie S(b,c) \bowtie T(c,d) \bowtie U(d,a)$选择最优的连接顺序

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

2个关系上的最优连接顺序

	$\{R,S\}$	$\{R,T\}$	$\{R,U\}$	$\{S,T\}$	$\{S,U\}$	$\{T,U\}$
估计基数	5,000	1M	10,000	2,000	1M	1,000
估计代价	0	0	0	0	0	0
最优计划	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$

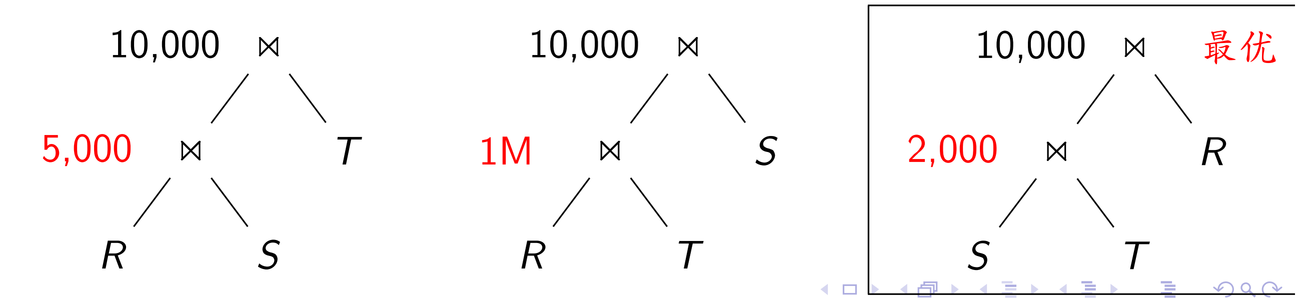
2个关系上的最优连接顺序

	$\{R,S\}$	$\{R,T\}$	$\{R,U\}$	$\{S,T\}$	$\{S,U\}$	$\{T,U\}$
估计基数	5,000	1M	10,000	2,000	1M	1,000
估计代价	0	0	0	0	0	0
最优计划	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$

3个关系上的最优连接顺序

	$\{R,S,T\}$	$\{R,S,U\}$	$\{R,T,U\}$	$\{S,T,U\}$
估计基数	10,000	50,000	10,000	2,000
估计代价	2,000	5,000	1,000	1,000
最优计划	$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(T \bowtie U) \bowtie S$

$R \bowtie S \bowtie T$的最优查询计划是如何得到的？

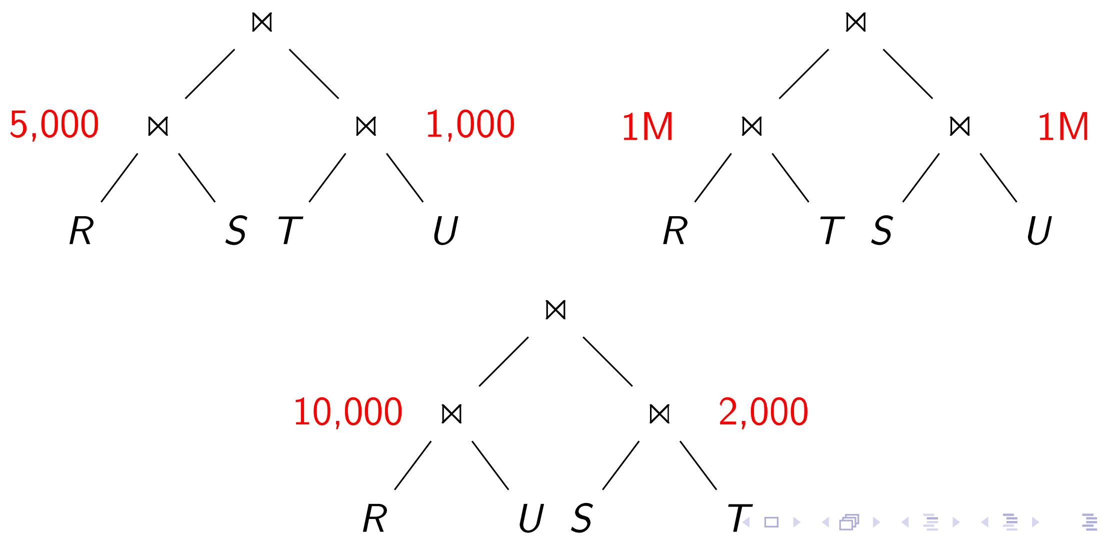


$R \bowtie S \bowtie T \bowtie U$的最优查询计划是如何得到的?

情况1: 连接顺序形如$(* \bowtie *) \bowtie (* \bowtie *)$

2个关系上的最优连接顺序

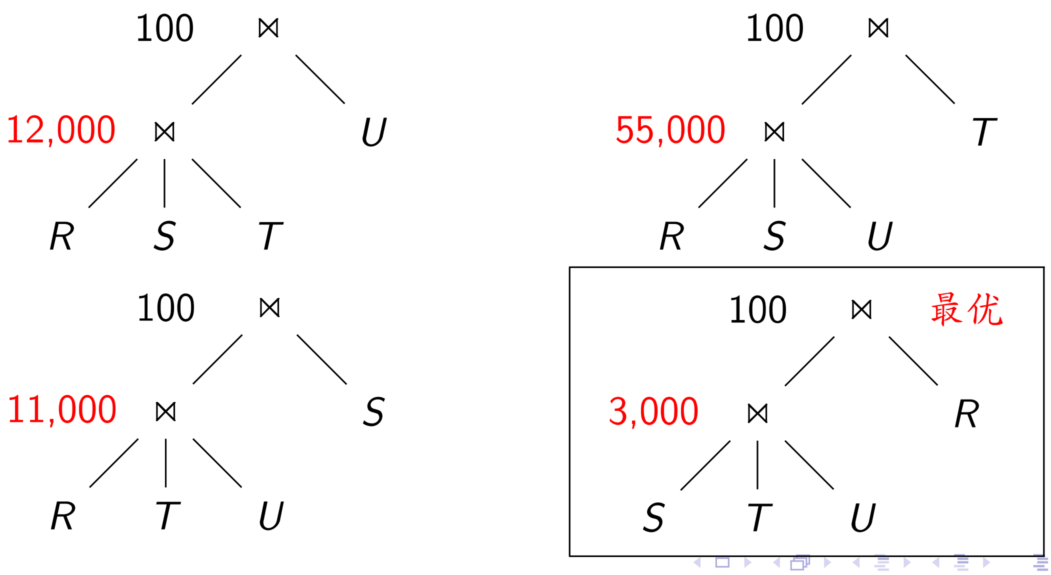
	$\{R,S\}$	$\{R,T\}$	$\{R,U\}$	$\{S,T\}$	$\{S,U\}$	$\{T,U\}$
估计基数	5,000	1M	10,000	2,000	1M	1,000
估计代价	0	0	0	0	0	0
最优计划	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$



情况2: 连接顺序形如$(* \bowtie * \bowtie *) \bowtie *$

3个关系上的最优连接顺序

	$\{R,S,T\}$	$\{R,S,U\}$	$\{R,T,U\}$	$\{S,T,U\}$
估计基数	10,000	50,000	10,000	2,000
估计代价	2,000	5,000	1,000	1,000
最优计划	$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(T \bowtie U) \bowtie S$



Improving Physical Query Plans

物理查询计划(Physical Query Plans)的优化

- **任务1:** 为逻辑查询计划中每个操作选择合适的物理执行算法
- **任务2:** 当把一个操作的结果传递给下一个操作时，为查询计划选择合适的执行模型

确定选择操作的物理执行算法

- 基于索引的选择(index-based selection)
- 基于索引的选择+求交集(index-merge)
- 基于索引的选择+过滤
- 基于扫描的选择(scanning-based selection)

基于索引的选择(Index-based Selection)

适用条件

- 选择条件的形式是$K = v$或$l \leq K \leq u$
- 关系R上建有属性K上的索引

方法：使用基于索引的选择算法

基于索引的选择+求交集(Index-Merge)

适用条件

- 选择条件的形式是$A = a \wedge B = b$
- 关系R上分别建有属性A和属性B上的索引

方法

- 分别在两个索引上找到满足条件$A = a$和$B = b$的元组地址集合
- 计算两个元组地址集合的交集
- 根据交集中的元组地址从文件中读取元组

基于索引的选择+过滤

适用条件

- 选择条件的形式是$A = a \wedge \theta$
- 关系R上建有属性A上的索引

方法

- 在索引上查找满足条件$A = a$的元组
- 对每个元组，再使用条件θ进行过滤

基于扫描的选择(Scanning-based Selection)

其他情况下使用基于扫描的选择算法

确定连接操作的执行方法

- 一趟连接(One-Pass Join)
- 索引连接(Index Join)
- 排序归并连接(Sort-Merge Join)
- 哈希连接(Hash Join)
- 嵌套循环连接(Nested-Loop Join)

一趟连接(One-Pass Join)

适用条件: 左关系可以全部读入缓冲池的可用页面

索引连接(Index Join)

适用条件

- 左关系较小
- 右关系在连接属性上建有索引

排序归并连接(Sort-Merge Join)

适用条件

- 至少有一个关系已经按连接属性排序
- 多个关系在相同连接属性上做多路连接也适合使用排序归并连接，如$R(a,b) \bowtie S(a,c) \bowtie T(a,d)$

哈希连接(Hash Join)

在一趟连接、排序归并连接、索引连接都不适用的情况下，哈希连接总是好的选择

嵌套循环连接(Nested-Loop Join)

当内存缓冲区的可用页面特别少时，可使用嵌套循环连接

查询计划的执行模型

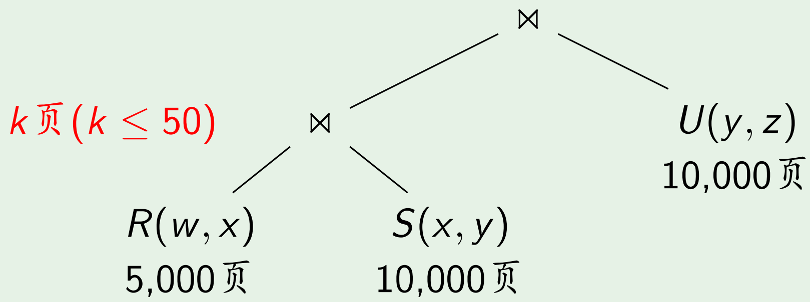
- **模型1:** 物化执行(materialization)
- **模型2:** 流水线执行(piplining)/火山模型(the volcano model)

缓冲池的可用页数影响着查询计划执行模型的选择

物理查询计划生成

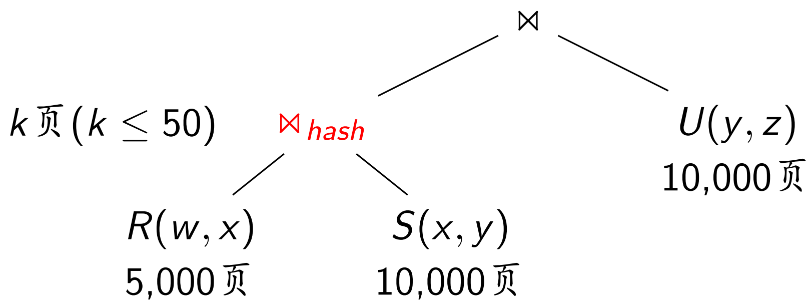
Example (物理查询计划生成)

逻辑查询计划



- 缓冲池中有$M = 101$个可用页
- R, S, U上均无索引且未按连接属性排序
- $R \bowtie S$的结果占$k \leq 50$页

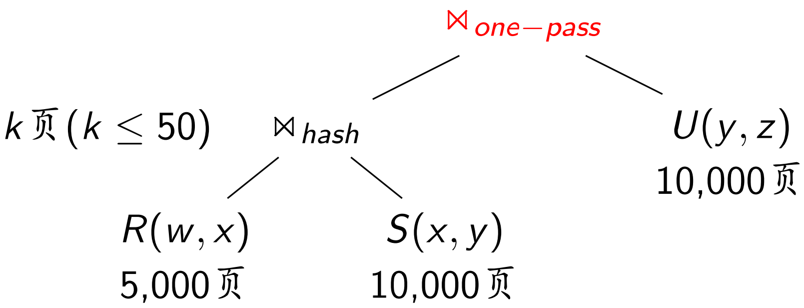
第1步



使用哈希连接(hash join)执行$R \bowtie S$

- 哈希分桶阶段使用101页内存
  - 输入缓冲 1页
  - 100个桶 100页
- 逐桶连接阶段使用51页内存(不计输出缓冲)
  - S的缓冲 1页
  - R的桶 50页
  - 输出缓冲 50页
- I/O代价：$3B(R) + 3B(S) = 45000$
- 因为$k \leq 50$，所以$R \bowtie S$的结果可以保留在输出缓冲区中，以流水线的形式输入给下一个连接操作

第2步



使用一趟连接(one-pass join)执行$(R \bowtie S) \bowtie U$

- 一趟连接使用$k+1$页内存(不计输出缓冲)

  U的缓冲 ▭ 1页

  $R \bowtie S$的结果 ▭ k页

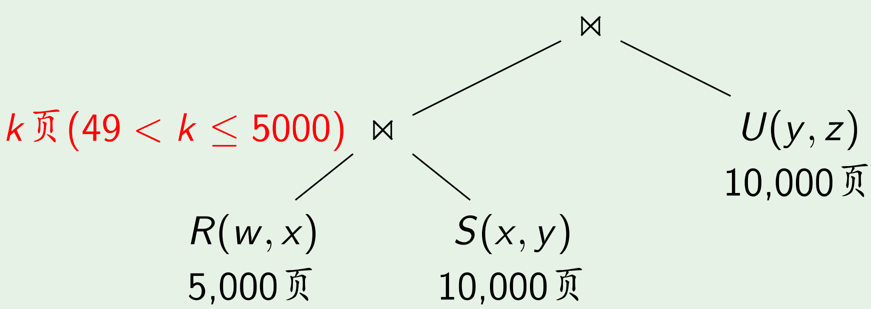
  输出缓冲 ▭ $100-k$页

- I/O代价：$B(U) = 10000$ ($R \bowtie S$的结果已在内存中，无需I/O)

制定物理查询计划

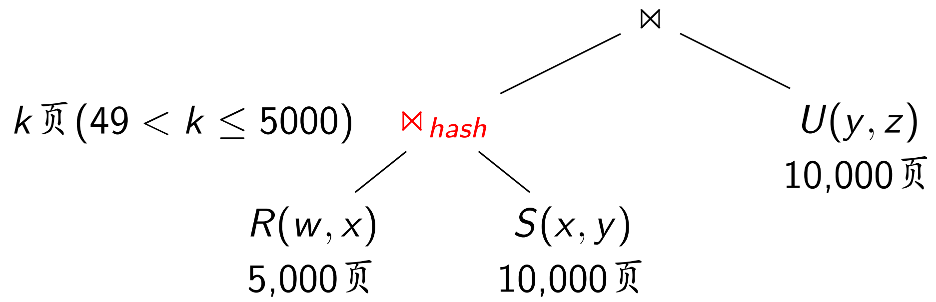
Example (确定物理查询计划)

逻辑查询计划



- 缓冲池中有$M = 101$个可用页
- R, S, U上均无索引且未按连接属性排序
- $R \bowtie S$的结果占$49 < k \leq 5000$块

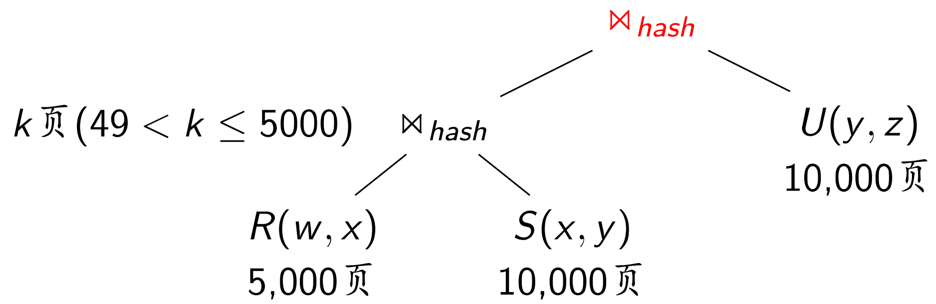
第1步



使用哈希连接(hash join)执行$R \bowtie S$

- 哈希分桶阶段使用101页内存
  输入缓冲 1页
  100个桶 100页
- 逐桶连接阶段使用51页内存(不计输出缓冲)
  S的缓冲 1页
  R的桶 50页
  输出缓冲 50页
- I/O代价: $3B(R) + 3B(S) = 45000$
- 因为$k > 50$，所以$R \bowtie S$的结果无法全部保留在输出缓冲区中，但仍以流水线形式输入给下一个连接操作

第2页



使用哈希连接(hash join)执行$(R \bowtie S) \bowtie U$

- $R \bowtie S$的结果哈希分桶阶段使用50页内存
  执行$R \bowtie S$ 51页
  50个桶 50页
- U的哈希分桶阶段使用51页内存
  U的缓冲 1页
  50个桶 50页
- 逐桶连接阶段使用101页内存(不计输出缓冲)
  U的缓冲 1页
  $R \bowtie S$的桶 100页
- I/O代价: $2B(R \bowtie S) + 3B(S) = 2k + 30000$

课后习题 I

1. 已知下列关系的统计信息，估计下面关系代数表达式结果的基数

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

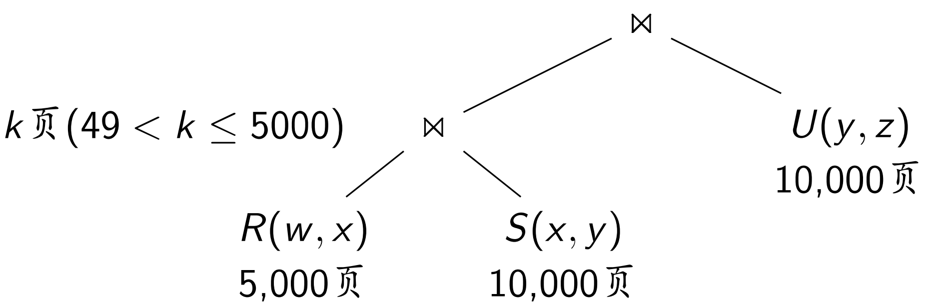
   1. $W \bowtie X \bowtie Y \bowtie Z$
   2. $\sigma_{a=10}(W)$
   3. $\sigma_{c=20}(Y)$
   4. $\sigma_{c=20}(Y) \bowtie Z$
   5. $W \times Y$
   6. $\sigma_{d>10}(Z)$
   7. $\sigma_{a=1 \wedge b=2}(W)$
   8. $\sigma_{a=1 \wedge b>2}(W)$
   9. $\sigma_{a<1 \wedge a>2}(W)$
   10. $X \bowtie_{X.c<Y.c} Y$

课后习题 II

2. 已知关系$R(a,b)$和$S(b,c)$的属性$R.b$和$S.b$的直方图如下，且$V(R,b)=V(S,b)=20$，估计$R \bowtie S$的结果大小

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

3. 已知逻辑查询计划如下



- 缓冲池中有$M=101$个可用页
- R, S, T上均无索引且未按连接属性排序
- $R \bowtie S$的结果占$49 < k \leq 5000$块

课后习题 III

假设我们使用哈希连接计算$R \bowtie S$，使用嵌套循环连接计算$(R \bowtie S) \bowtie U$。分析该查询计划的I/O代价和内存使用情况

Summary

1. Overview
2. Improving Logical Query Plans
   - Transformations of Relational Algebra Expressions
   - Estimation of Query Plan Cost
   - Optimization of Join Orders
3. Improving Physical Query Plans