

# 第八章 查询优化

**章成源**

湖南大学-信息科学与工程学院-计算机科学系

办公室：院楼403

Email: [cyzhangcse@hnu.edu.cn](mailto:cyzhangcse@hnu.edu.cn)

# 第八章 查询优化

## 8.1 查询优化概述

## 8.2 关系代数表达式等价变换规则

## 8.3 统计信息

## 8.4 基数估算

## 8.5 多表连接的优化

## 8.6 小结

# 第八章 查询优化

**8.1.1 查询优化的意义**

**8.1.2 查询优化方法**

**8.1.3 查询优化器的结构**

## 8.1.1 查询优化的意义

- 查询优化既是关系数据库管理系统实现的关键技术，它减轻了用户选择存取路径以及如何更好地表达查询以获得较高的效率的负担。
- 查询优化器的主要优点：
  - 根据数据字典中的统计信息做出正确的估算，选择高效的执行计划
  - 优化器可以考虑千百种不同的执行计划
  - 优化器中包括很多复杂的优化技术并提供给所有用户

## 8.1.1 查询优化的意义

- 示例（为什么需要查询优化）：
- 查找订单额在50000元以上的订单和顾客信息，用SQL语句表达：

```
SELECT CName, Amount
```

```
FROM Customers C, Orders O
```

```
WHERE O.CID = C.CID AND Amount > 50000;
```

## 8.1.1 查询优化的意义

- 示例（续）：
  - 查假定数据库中有1000个顾客记录，10 000个订单记录，其中订单额大于50000的订单数记录为50个，三种等价的关系代数表达式如下：

$$Q_j = \Pi_{CName \parallel Amount}(\sigma_{OgCID=CgCID \wedge Amount > 50000}(Customers \times Orders))$$

$$Q_k = \Pi_{CName \parallel Amount}(\sigma_{Amount > 50000}(Customers \bowtie Orders))$$

$$Q_l = \Pi_{CName \parallel Amount}(Customers \bowtie \sigma_{Amount > 50000}(Orders))$$

## 8.1.1 查询优化的意义

- 示例（续）：
  - Q1先进行两个表的笛卡儿积，产生 $10^7$ 个元组，然后进行选择 and 投影操作；
  - Q2进行自然连接，生成10000个元组，然后进行选择 and 投影；
  - Q3先进行选择操作，只选出满足条件的50个元组，然后进行连接 and 投影运算；
  - 显然，Q1由于中间结果太大，导致中间结果I/O开销大，从执行时间上看，Q1会远远大于Q2和Q3，Q3具有最优的执行时间。

## 8.1.2 查询优化方法

- 逻辑查询优化
  - 关系代数表达式的优化，即按照一定的规则，通过对关系代数表达式进行等价变换，改变代数表达式中操作的次序和组合，使查询执行更高效，不涉及底层的存取路径。

## 8.1.2 查询优化方法

- 物理查询优化
  - 给每个关系运算符选择高效合理的运算方法或存取路径，生成优化的可执行的查询计划。
  - 主要考虑：存取路径；多表连接顺序；表连接算法；操作结果是否排序，数据传递方式等。

## 8.1.2 查询优化方法

- 查询优化器选择执行计划的方法
  - 基于规则的启发式优化（RBO）
  - 基于代价估算的优化（CBO）
  - 基于机器学习的查询优化（ABO）

## 8.1.2 查询优化方法

- 基于规则的启发式优化（RBO）
  - 指根据预定义的启发式规则对SQL语句进行优化；
  - 常见的启发式规则：选择投影运算尽量下推、多表连接时尽量让小表先进行连接等。
  - 假定因素太多，准确程度相对较差，一般不单独使用，目前的系统通常使用启发式规则来减少候选执行计划。

## 8.1.2 查询优化方法

- 基于代价估算的优化（CBO）

- 使用优化器估算不同执行策略的代价，并选出具有最小代价的执行计划。

- 关键技术：

- （1）执行路径的代价估算，包括统计信息的准确收集、合理的代价模型以及对中间结果的估算方法等。

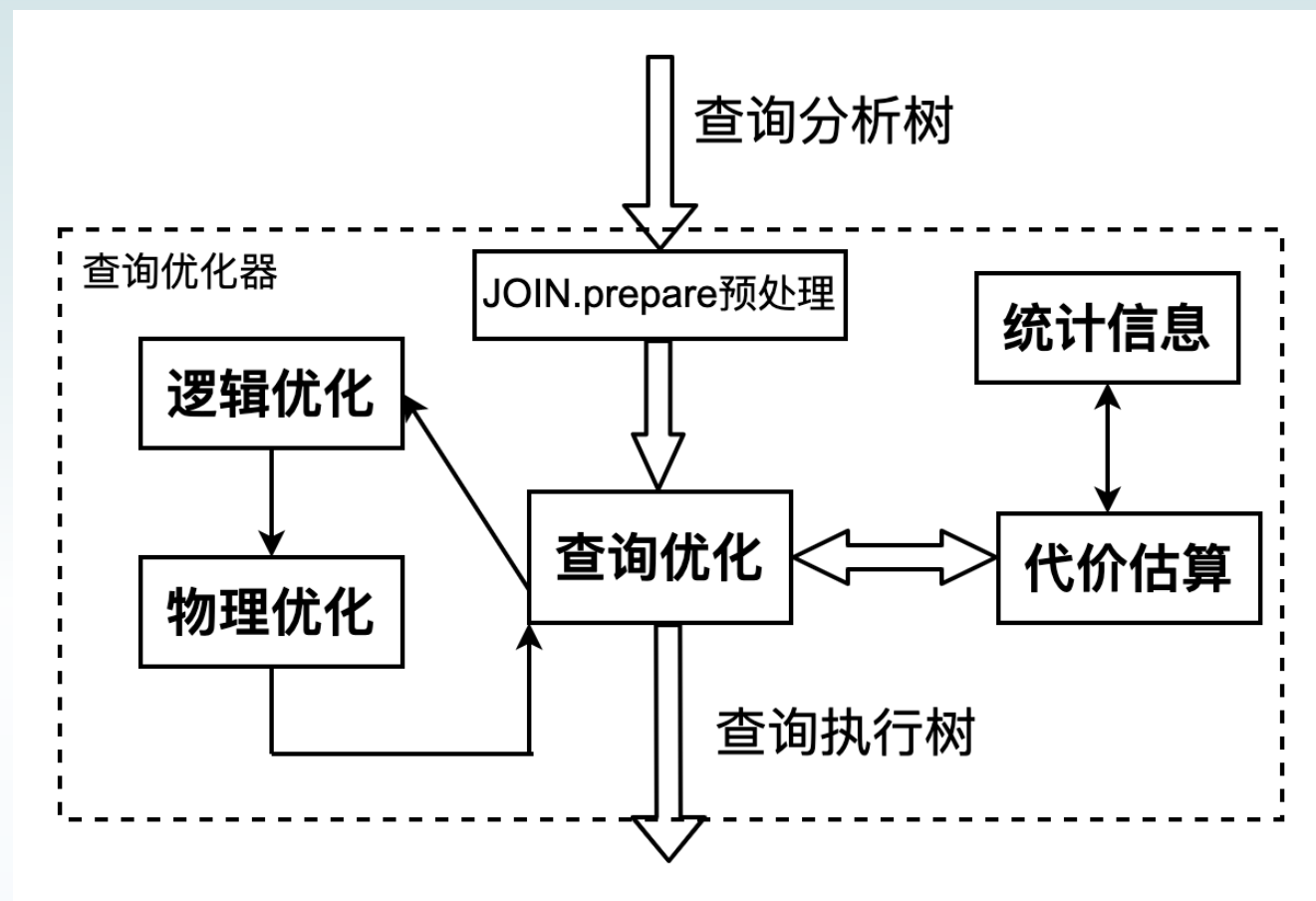
- （2）执行路径的生成以及最优路径选择，如何从指数级增长的执行路径中高效选取代价最小的执行路径。

## 8.1.2 查询优化方法

- 基于机器学习的查询优化（ABO）
  - ABO收集执行计划的特征信息，借助机器学习模型获得经验信息，进而对执行计划进行调优，获得最优的执行计划。
  - 在建模效率、估算准确率、自适应等方面有很大优势。

## 8.1.3 查询优化器的结构

- 查询优化器
  - 输入：查询分析树
  - 输出：查询执行树
  - 两类经典查询优化器：
    - SystemR/Starburst
    - Volcano/Cascades



## 8.1.3 查询优化器的结构

- 查询优化器SystemR/Starburst
  - 逻辑优化阶段：基于启发式规则进行查询重写，把查询代数表达式变换成另一个等价的执行更高效的查询代数表达式；
  - 物理优化阶段：给查询树中的关系代数表达式关联物理实现的操作符，评估其代价，选择最优的执行路径，采用自底向上的方式生成生成物理执行计划.。

## 8.1.3 查询优化器的结构

- 查询优化器SystemR/Starburst
  - 优点：层次清晰
  - 缺点：逻辑优化与物理优化完全独立，在逻辑优化阶段，有些规则不能判断其一定有什么好处，需要计算代价来判断时，通过启发式规则不能获得最优解。

## 8.1.3 查询优化器的结构

- 查询优化器Volcano/Cascades
  - 事先定义了Operator、Property和Rule的接口类，通过实现这些接口类的子类可以实现一个查询优化器。
  - 使用规则进行所有的关系代数变换：
    - 转换规则（transformation rules）进行关系代数的等价变换；
    - 实现规则（implementation rules）把关系代数表达式转换成操作符树。

## 8.1.3 查询优化器的结构

- 查询优化器Volcano/Cascades
  - 所有的转换都是基于规则和代价。
  - 采用自顶向下的动态规划算法（Memorization）搜索状态空间，保证只有需要优化的子树和相关的属性才加入到查询计划中；
  - 采用MEMO结构尽可能共享相同的子树以减少内存的使用。

## 8.1.3 查询优化器的结构

- 查询优化器Volcano/Cascades
  - 优点：统一使用规则来生成转换，具有更好的扩展性；
  - 缺点：系统结构复杂

# 第八章 查询优化

8.1 查询优化概述

8.2 关系代数表达式等价变换规则

8.3 统计信息

8.4 基数估算

8.5 多表连接的优化

8.6 小结

# 第八章 查询优化

8.2.1 选择运算的相关规则

8.2.2 投影运算的相关规则

8.2.3 连接运算的相关规则

8.2.4 去重运算的相关规则

8.2.5 聚集运算的相关规则

8.2.6 集合运算的相关规则

## 8.2 关系代数表达式等价变换规则

- 逻辑查询优化主要通过对关系代数表达式进行等价变换来提高查询效率；
- 关系代数表达式的等价：指用相同的关系代替两个表达式中相应的关系所得到的结果是相同的。
- 两个关系表达式 $E_j$ 和 $E_k$ 是等价的，可记为 $E_j = E_k$ 。

## 8.2.1 选择运算的相关规则

- 规则1：选择的分解规则
  - 设 $E_j$ 和 $E_k$ 是关系代数表达式， $F_j$ 和 $F_k$ 是选择运算的谓词，则有：

$$\sigma_{F_j \wedge F_k}(E) = \sigma_{F_j}(\sigma_{F_k}(E)) = \sigma_{F_k}(\sigma_{F_j}(E))$$

$$\sigma_{F_j \vee F_k}(E) = \sigma_{F_j}(E) \cup \sigma_{F_k}(E)$$

## 8.2.1 选择运算的相关规则

- 例1：对于关系 $R(a,b,c)$ ， $R$ 在属性 $a$ 上有索引
  - 对于 $\sigma_{a=1 \wedge b=1}(R)$ 可以转化为 $\sigma_{b=1}(\sigma_{a=1}(R))$
  - 首先利用属性 $a$ 上的索引进行查找，然后根据条件 $b=1$ 进行选择操作，得出满足条件的结果集。

## 8.2.1 选择运算的相关规则

- 例2：对于关系 $R(a,b,c)$ ， $R$ 在属性 $a$ 和 $b$ 上有索引
  - 对于 $\sigma_{a=1 \vee b=1}(R)$ 进行选择操作时可能需要进行全表扫描；
  - 若转化为 $\sigma_{a=1}(R) \cup \sigma_{b=1}(R)$ ，则可以分别利用属性 $a$ 和属性 $b$ 上的索引进行扫描，然后使用集合操作把结果集合并。

## 8.2.1 选择运算的相关规则

- 规则2：选择与笛卡儿积、连接运算的分配律
  - 设 $E_j$ 和 $E_k$ 是关系代数表达式， $F$ 是选择运算的谓词， $\theta$ 是连接运算的谓词，根据 $\theta$ 连接的定义有：

$$\sigma_{\theta}(E_j \times E_k) = E_j \bowtie_{\theta} E_k$$

$$\sigma_{\theta_1}(E_j \bowtie_{\theta_2} E_k) = E_j \bowtie_{\theta_1 \wedge \theta_2} E_k$$

## 8.2.1 选择运算的相关规则

- 对于 $\theta$ 连接，选择运算在下面两种情况下满足分配律：

(1) 如果 $F$ 中涉及的属性都是 $E_j$ 中的属性，则

$$\sigma_F(E_j \bowtie_{\theta} E_k) = \sigma_F(E_j) \bowtie_{\theta} E_k$$

(2) 如果 $F = F_j \wedge F_k$ ,  $F_j$ 只涉及 $E_j$ 中的属性， $F_k$ 只涉及 $E_k$ 中的属性，则

$$\sigma_F(E_j \bowtie_{\theta} E_k) = \sigma_{F_i}(E_j) \bowtie_{\theta} \sigma_{F_j}(E_k)$$

## 8.2.1 选择运算的相关规则

- 对于 $\theta$ 连接，选择运算在下面两种情况下满足分配律：

(3)  $F_j$  只涉及  $E_j$  中的属性， $F_k$  涉及  $E_j$  和  $E_k$  两者的属性，则

$$\sigma_F(E_j \bowtie_{\theta} E_k) = \sigma_{F_k}(\sigma_{F_j}(E_j) \bowtie_{\theta} E_k)$$

使用该规则可以在连接前先做部分选择操作，把选择操作下推到连接操作下面，从而减少参与连接运算的关系规模。

## 8.2.1 选择运算的相关规则

- 例： 关系 $R(a,b)$ 和 $S(b,c)$ ,对于表达式 $\sigma_{(a=1 \vee a=3) \wedge c < 10} (R \bowtie S)$ 
  - 其含义是 $R$ 与 $S$ 做自然连接，然后再做选择操作。
  - $R$ 与 $S$ 的自然连接有可能产生比较大的中间结果集。
  - 条件  $(a=1 \vee a=3)$  只作用到关系 $R$ 上，条件 $b < c$ 只作用在关系 $S$ 上。

## 8.2.1 选择运算的相关规则

- 例（续）：
- 则根据规则，表达式可以变换如下：

$$\sigma_{(a=j \vee a=l) \wedge c < j i} (R \bowtie S)$$

$$= \sigma_{(a=j \vee a=3)} (R) \bowtie \sigma_{(c < j i)} (S)$$

- 即 通过变换把选择谓词下推到关系上，先进行选择操作，再进行连接操作。

## 8.2.1 选择运算的相关规则

- 规则3：选择与集合操作的分配律

- 若 $E_j$ 和 $E_k$ 有相同的属性名，则

(1) 对于集合并（ $\cup$ ）操作，选择操作下推到集合的两个参数中

$$\sigma_F(E_j \cup E_k) = \sigma_F(E_j) \cup \sigma_F(E_k)$$

(2) 对于集合并（ $\cap$ ）操作，选择操作下推到集合的一个或两个参数中

$$\begin{aligned}\sigma_F(E_j \cap E_k) &= \sigma_F(E_j) \cap E_k = E_j \cap \sigma_F(E_k) \\ &= \sigma_F(E_j) \cap \sigma_F(E_k)\end{aligned}$$

## 8.2.1 选择运算的相关规则

- 规则3：选择与集合操作的分配律

(3) 对于集合差（ $-$ ）操作，选择操作必须下推到集合的第一个参数中，下推到第二个参数是可选的：

$$\sigma_F(E_j - E_k) = \sigma_F(E_j) - E_k = \sigma_F(E_j) - \sigma_F(E_k)$$

- 通过上述规则，可以把选择操作下推到集合操作下面，从而减少参与集合运算的关系的规模。

## 8.2.2 投影运算的相关规则

- 规则4：投影的串接规则

$$\Pi_{A_1, A_2, \dots, A_n}(\Pi_{B_1, B_2, \dots, B_m}(E)) = \Pi_{A_1, A_2, \dots, A_n}(E)$$

- E为关系代数表达式， $A_i (i = 1, 2, \dots, n)$ ,  $B_j (j = 1, 2, \dots, m)$ 为属性名，且  $\{A_1, A_2, \dots, A_n\}$  构成  $\{B_1, B_2, \dots, B_m\}$  的子集。

## 8.2.2 投影运算的相关规则

- 规则5：投影与选择运算的交换

$$\sigma_F(\Pi_{A_1 \leftarrow A_2 \leftarrow \dots \leftarrow A_n}(E)) = \Pi_{A_1 \leftarrow A_2 \leftarrow \dots \leftarrow A_n}(\sigma_F(E))$$

- $F$  只涉及属性  $A_1, A_2, \dots, A_n$ ，若  $F$  中不属于  $A_1, A_2, \dots, A_n$  的属性  $B_1, B_2, \dots, B_m$ ，则有更一般的规则：

$$\Pi_{A_1 \leftarrow A_2 \leftarrow \dots \leftarrow A_n}(\sigma_F(E)) = \Pi_{A_1 \leftarrow A_2 \leftarrow \dots \leftarrow A_n}(\sigma_F(\Pi_{A_1 \leftarrow A_2 \leftarrow \dots \leftarrow A_n \leftarrow B_1 \leftarrow B_2 \leftarrow \dots \leftarrow B_m}(E)))$$

## 8.2.2 投影运算的相关规则

- 规则6: 投影与笛卡尔积、连接操作的分配律

设 $E_j$ 和 $E_k$ 是两个关系表达式,  $A_1, A_2, \dots, A_n$ 是 $E_j$ 的属性子集,  $B_1, B_2, \dots, B_m$ 是 $E_k$ 的属性子集, 则投影操作对于笛卡尔积满足分配律:

$$\Pi_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}(E_j \times E_k) = \Pi_{A_1, A_2, \dots, A_n}(E_j) \times \Pi_{B_1, B_2, \dots, B_m}(E_k)$$

## 8.2.2 投影运算的相关规则

- 对于 $\theta$ 连接，投影在下面两种情况下满足分配律：

(1) 如果 $\theta$ 中涉及的属性都是 $\{A_j, A_k, \dots, A_n, B_j, B_k, \dots, B_m\}$ 中的属性，则的属性是 $E_j$ 的属性子集， $B_j, B_k, \dots, B_m$ 是 $E_k$ 的属性子集,则投影操作对于笛卡尔积满足分配律：

$$\Pi_{A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m}(E_j \times E_k) = \Pi_{A_1, A_2, \dots, A_n}(E_j) \times \Pi_{B_1, B_2, \dots, B_m}(E_k)$$

## 8.2.2 投影运算的相关规则

- 对于 $\theta$ 连接，投影在下面两种情况下满足分配律：

(2) 如果 $\theta$ 中涉及 $E_j$ 中不在 $\{A_j, A_k, \dots, A_n\}$ 中的属性 $\{C_j, \dots, C_i\}$ , 和 $E_k$ 中不在 $\{B_j, B_k, \dots, B_m\}$ 中的属性 $\{D_j, \dots, D_k\}$ , 则：

$$\begin{aligned} & \Pi_{A_1, A_2, \dots, A_n, B_1 \in B_1 \text{ eggge } B_m} (E_j \bowtie_{\theta} E_k) \\ &= \Pi_{A_1, A_2, \dots, A_n} (\Pi_{A_1, A_2, \dots, A_n, C_1 \dots C_i} (E_j) \bowtie_{\theta} \Pi_{B_1 \in B_1 \text{ eggge } B_m \in D_1 \text{ eggge } D_k} (E_k)) \end{aligned}$$

## 8.2.2 投影运算的相关规则

- 规则7：投影与集合操作的分配律

设 $E_j$ 和 $E_k$ 有相同的属性名， $A_j, A_k, \dots, A_n$ 是其属性，则：

$$\Pi_{A_j, A_k, \dots, A_n}(E_j \cup E_k) = \Pi_{A_j, A_k, \dots, A_n}(E_j) \cup \Pi_{A_j, A_k, \dots, A_n}(E_k)$$

- 可以在查询表达式树的任何地方引入投影，只要它所消除的属性是后面的运算符从来不会用到并且也没有出现在整个表达式的结果之中。

## 8.2.3 连接运算的相关规则

- 交换律和结合律
  - 运算符的交换律：指该运算符的运算结果与其参数顺序无关。
  - 运算符的结合律：指该运算符出现的两个地方既可以从左边进行组合也可以从右边进行组合。
  - 当一个运算符既满足交换律又满足结合律时，就可以对这个运算符连接起来的任意多个操作数进行随意组合和排列而不会影响运算结果。

## 8.2.3 连接运算的相关规则

- 规则8：连接、笛卡儿积的交换律

设 $E_j$ 和 $E_k$ 是关系代数表达式， $\theta$ 是连接运算的谓词，则：

$$E_j \times E_k = E_k \times E_j$$

$$E_j \bowtie E_k = E_k \bowtie E_j$$

$$E_j \bowtie_{\theta} E_k = E_k \bowtie_{\theta} E_j$$

## 8.2.3 连接运算的相关规则

- 规则9：连接、笛卡儿积的结合律

设 $E_1$ 、 $E_2$ 、 $E_3$ 是关系代数表达式， $\theta_1$ 、 $\theta_2$ 是连接运算的谓词，则：

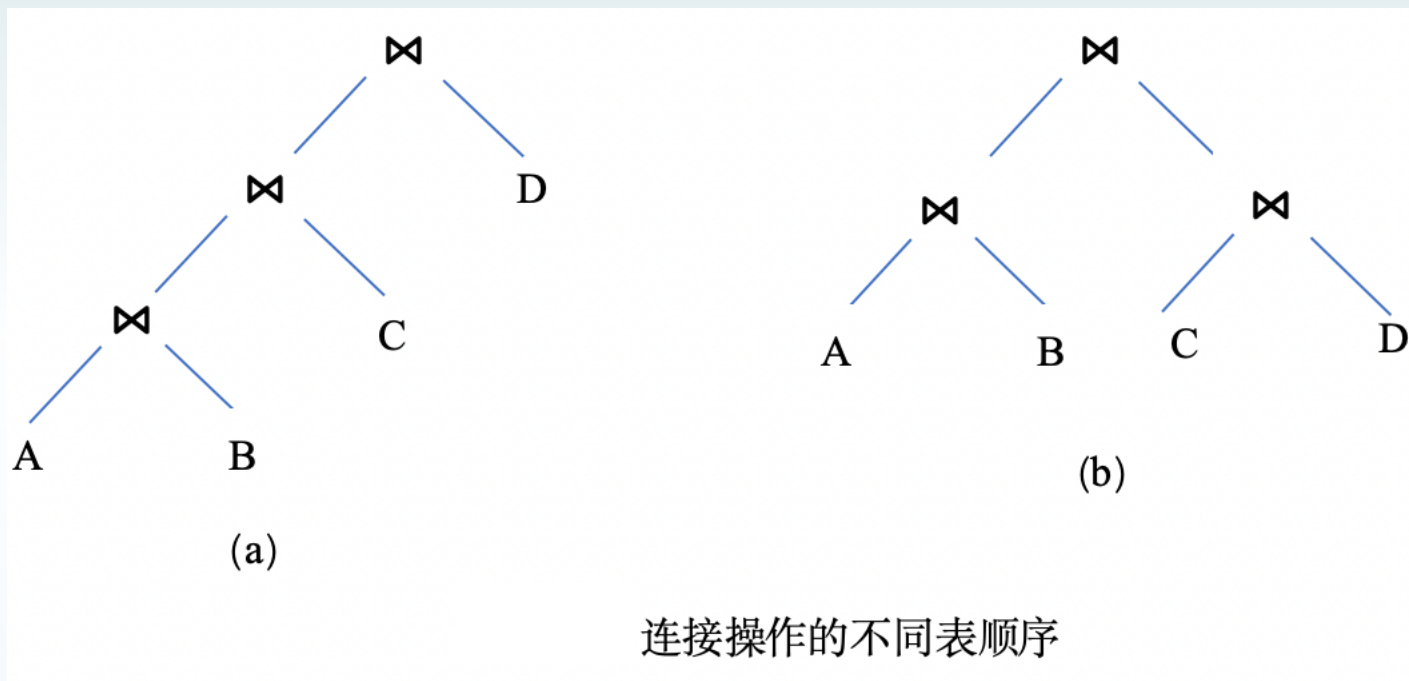
$$(E_1 \times E_2) \times E_3 = E_1 \times (E_2 \times E_3)$$

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

$$(E_1 \theta_1 E_2) \theta_2 E_3 = E_1 \theta_1 (E_2 \theta_2 E_3)$$

## 8.2.3 连接运算的相关规则

- 例：假设有A、B、C、D四个表进行连接，则连接方式可以是线性的  
 $((A \bowtie B) \bowtie C) \bowtie D$ ，也可以是非线性的 $(A \bowtie B) \bowtie (C \bowtie D)$



## 8.2.3 连接运算的相关规则

- 例：运用关系代数表达式等价规则进行查询转换，其中SQL语句查询顾客 Smith在2020年的订单明细信息：

```
SELECT CName, PID, Quantity
FROM Customers C , Orders O, OrderItems I
WHERE C.CID = O.CID
      AND O.OID = I.OID
      AND C.CName = 'Smith'
      AND date_part('year', CreateTime) = 2020;
```

## 8.2.3 连接运算的相关规则

- 例(续): 该SQL查询对应的关系代数表达式如下所示

$\Pi_{CName, PID, Quantity}(\sigma_{CName='Smith' \wedge date\_part('year', CreateTime)=2020}(Customer \bowtie Orders \bowtie OrderItems))$

- 对于该查询通常做的转换是下推选择操作、投影操作来减少中间结果集的规模，关系代数表达式的转换如下所示:

$\Pi_{CName, PID, Quantity}(\sigma_{CName='Smith' \wedge date\_part('year', CreateTime)=2020}(Customer \bowtie Orders \bowtie OrderItems))$   
=  $\Pi_{CName, PID, Quantity}((\sigma_{CName='Smith' \wedge date\_part('year', CreateTime)=2020}(Customer \bowtie Orders \bowtie OrderItems))$   
=  $\Pi_{CName, PID, Quantity}((\sigma_{CName='Smith'}(Customers) \bowtie \sigma_{date\_part('year', CreateTime)=2020}(Orders)) \bowtie OrderItems))$   
=  $\Pi_{CName, PID, Quantity}(\Pi_{CName, CID}(\sigma_{CName='Smith'}(Customers)) \bowtie \Pi_{CID, OID}(\sigma_{date\_part('year', CreateTime)=2020}(Orders)) \bowtie \Pi_{OID, PID, Quantity}(OrderItems))$

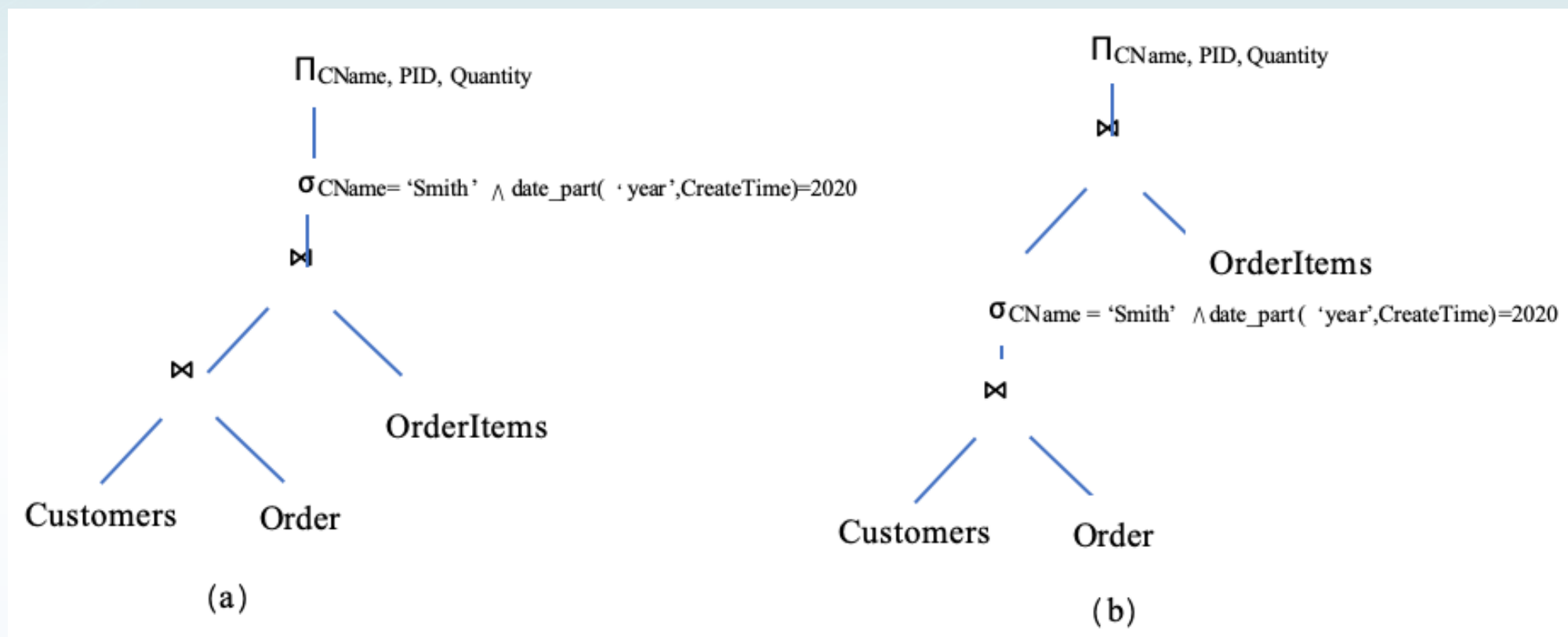
连接的结合律

下推选择操作

下推投影操作

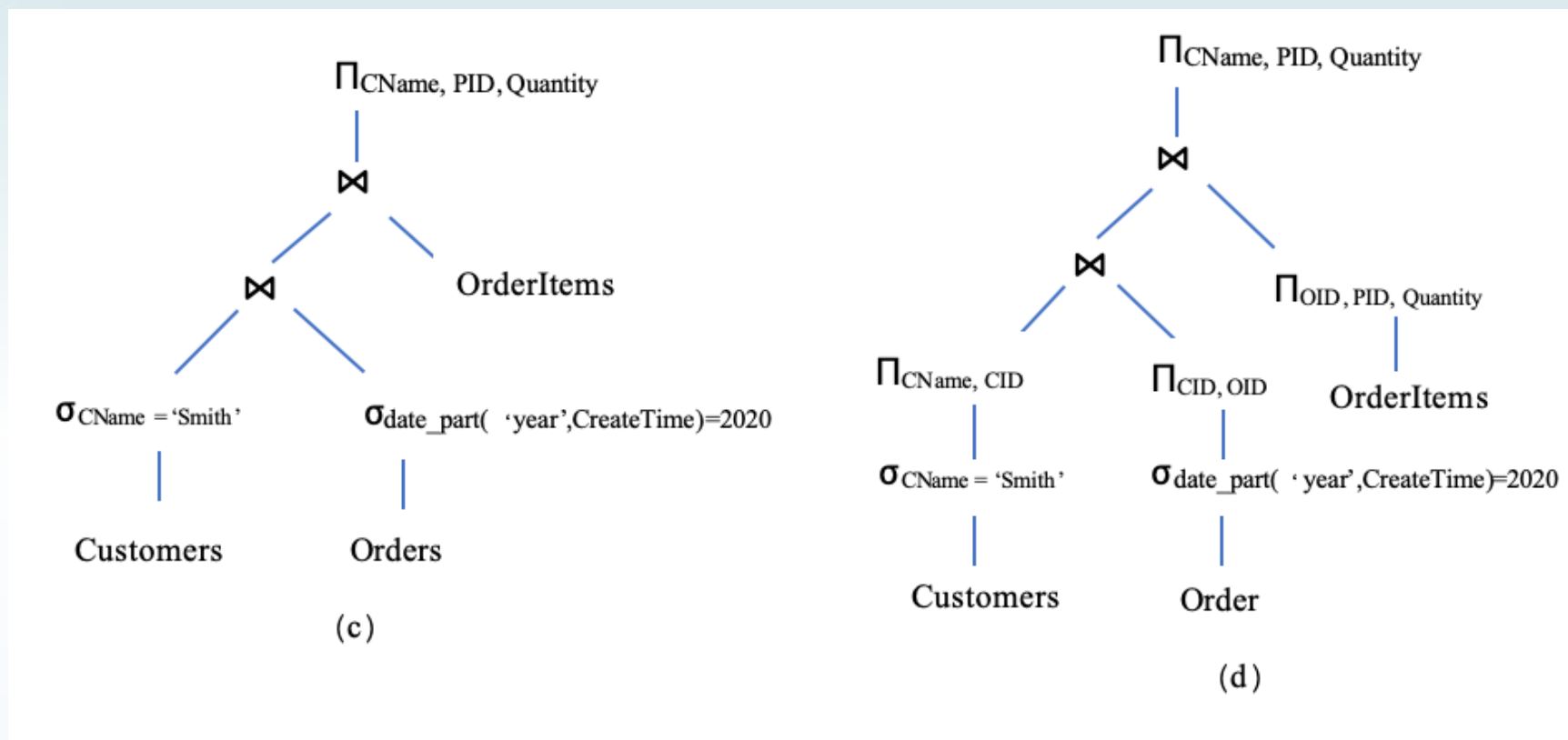
## 8.2.3 连接运算的相关规则

- 例(续): 上述表达式变换的等价关系代数表达式树表示如图:



## 8.2.3 连接运算的相关规则

- 例(续): 上述表达式变换的等价关系代数表达式树表示如图:



## 8.2.3 连接运算的相关规则

- 规则10：连接消除
  - 连接消除条件：如果参与连接运算是基于参照完整性的主外表，并且连接谓词是主外码的等值连接，则对于外表来说，连接运算的结果集不会增加其元组行数，如果主表只在连接中出现，则该连接就是冗余，主表消除后的查询与原查询等价。

## 8.2.3 连接运算的相关规则

- 例：数据库中有Suppliers和Products表，分别存储供应商和商品的信息，Suppliers和Products表之间有参照完整性约束，Products表中的字段SID参照Suppliers表的主码SID。现在有视图prod\_supp\_info，可以查询商品的供应商信息：

```
CREATE VIEW prod_supp_info AS  
  
SELECT PID, PName, Category, SName, City  
  
FROM Products P, Suppliers S  
  
WHERE S.SID = P.PID;
```

## 8.2.3 连接运算的相关规则

- 例（续）查询Q2中用户想查询商品的信息，可能由于访问权限的原因只能查询视图

prod\_supp\_info:

Q 2:

```
SELECT PID, PName FROM prod_supp_info;
```

在查询重写阶段，该查询中的视图替换成对基表的查询，如Q3所示：

Q 3:

```
SELECT PID, PName FROM Products P, Suppliers S WHERE S.SID = P.PID;
```

## 8.2.3 连接运算的相关规则

- 例（续）由于Products 和Suppliers表在连接列上有主外码参照关系，并且主码表Suppliers只在连接子句中出现，因此，Q3与下面消除连接后的查询Q4等价：

Q4:

```
SELECT PID, PName FROM Products WHERE SID is not null;
```

注：Q4中增加了WHERE 子句 SID is not null，因为外码列的值是空值的元组在连接运算中不满足连接谓词，这些元组不会出现在结果集中，所以消除连接后的查询需要增加相应的选择谓词。

## 8.2.3 连接运算的相关规则

- 由上例可知，连接消除的本质是参与连接的表有如下特点：
  - (1) 参与连接的表只出现在该连接操作中，SQL的其它子句未引用该表的列。
  - (2) 连接谓词中该表的列具有唯一性（通过连接不会增加元组的行数）。
    - 对于内连接，消除连接后需要对另一个表的连接列增加is not null谓词；
    - 对于外连接，则直接消除连接的表即可。

## 8.2.3 连接运算的相关规则

- 连接谓词中该表列的唯一性判断方法：
  - (1) 该表的列是主码或有unique索引。
  - (2) 如果参加连接的是子查询，则可以判断子查询中是否具有Distinct关键字、分组聚集运算的结果，或者是集合并、交、差运算的结果（不含union all）。

## 8.2.4 去重运算的相关规则

- 去重运算符用于去掉集合中的重复元组，通常采用排序或哈希算法实现，是比较耗时的运算符。
- 它同时可以减少中间结果集的规模，因此将去重运算符下推可能获益。

## 8.2.4 去重运算的相关规则

- 规则11：消除去重运算符
  - 若关系E没有重复元组，则

$$\delta(E) = E$$

- 关系E可以是：包含主码、或者是分组运算的结果、或者是集合并、交、差运算的结果（不含union all）。

## 8.2.4 去重运算的相关规则

- 例：Distinct列上如果有主码约束，则此列不可能为空，且无重复值，可以消除 Distinct操作：

```
create table T1( c1 int primary key, c2 int);
```

```
SELECT DISCINCT(c1) FROM T1 ;
```

可以转换为：

```
SELECT c1 FROM T1 ;
```

## 8.2.4 去重运算的相关规则

- 规则12：去重与笛卡尔积、连接的分配律
  - 设 $E_1$ 和 $E_2$ 是关系代数表达式，则有

$$\delta_{(E_1 \times E_2)} = \delta_{(E_1)} \times \delta_{(E_2)}$$

$$\delta_{(E_1 \bowtie E_2)} = \delta_{(E_1)} \bowtie \delta_{(E_2)}$$

## 8.2.4 去重运算的相关规则

- 规则13：去重与选择的交换
  - 设E是关系代数表达式，F是选择谓词，则有

$$\delta(\sigma_F(E)) = \sigma_F(\delta(E))$$

## 8.2.4 去重运算的相关规则

- 规则14：去重与集合的分配律

去重操作符可以移到集合交操作的其中一个或两个参数上，但是不适用于集合的并、差操作。设 $E_j$ 和 $E_k$ 是关系代数表达式，则有

$$\delta_{aE_j \cap E_k, b} = \delta_{aE_j, b} \cap \delta_{aE_k, b} = \delta_{aE_k, b} \cap E_j = E_j \cap \delta_{aE_k, b}$$

## 8.2.5 聚集运算的相关规则

- 标量聚集
  - 没有分组列，即没有GroupBy子句，无论其进行聚集操作的基本表中是否有数据，其查询结果集中总是返回1条元组。
- 向量聚集
  - 具有分组列，即具有Group By子句，当其进行聚集操作的基表中没有数据时，查询结果集返回0条元组，否则，查询结果的大小取决于分组列中值的分布。

## 8.2.5 聚集运算的相关规则

- 标量聚集：用  $\zeta_{jeF}$  表示
- 向量聚集：用  $\zeta_{AeF}$  表示
- 其中F代表要计算的聚集函数，F中涉及的属性列称为聚集列，A代表分组列。标量聚集可以看作是向量聚集的特殊形式： $\zeta_{jeF} = \zeta_{\phi eF}$
- 对于空集上的聚集运算，标量聚集返回一行结果集，而向量聚集则不返回结果集。

## 8.2.5 聚集运算的相关规则

- 规则15：聚集运算中聚集列的消减

若聚集操作中的聚集列中包含唯一键，则该聚集操作的聚集列只需要保留该唯一键即可，从而减少聚集操作的聚集列，进一步减少聚集操作的开销。

- 设 $E$ 是关系代数表达式， $U$ 是 $E$ 上的一个唯一键， $U \subseteq A$ ，则有

$$\zeta_{AeF}(E) = \zeta_{UeF}(E)$$

## 8.2.5 聚集运算的相关规则

- 规则16：聚集与去重
- 聚集操作的结果集在聚集列上是唯一的，因此，聚集操作后的去重操作可以消除。设E是关系代数表达式，则有：

$$\delta_{a\mathcal{G}_{A|F}aE}bb = \mathcal{G}_{AeF}(E)$$

- 有时可以把去重运算看作是聚集运算的特殊情况，即去重运算是没有聚集函数的聚集运算。

## 8.2.5 聚集运算的相关规则

- 规则16：聚集与去重(续)
- 如果聚集函数F是max( )或min( ),则该聚集操作的结果跟操作对象是否有重复值无关，对这类聚集函数，可以消除该操作对象上的去重操作。设E是关系代数表达式，则有：

$$\mathcal{G}_{A,F}(\delta_{(E)}) = \mathcal{G}_{A,F}(E)$$

## 8.2.5 聚集运算的相关规则

- 规则17：聚集与投影
- 在聚集操作之前，可以使用投影操作去除操作对象中无用的属性。
- 设E是关系代数表达式，M是至少包含A中涉及所有属性的列表，则有：

$$\zeta_{AeF}(E) = \zeta_{AeF}(\Pi_M(E))$$

## 8.2.5 聚集运算的相关规则

- 规则18：聚集与选择的交换
- 设E是关系代数表达式，P是选择谓词，如果P涉及列是分组列A的子集，则有：

$$\zeta_{A \in F}(\sigma_P(E)) = \sigma_P(\zeta_{A \in F}(E))$$

## 8.2.5 聚集运算的相关规则

- 例：Q5查询2022年的销售总额，其中的HAVING子句中的谓词只涉及到查询的分组列，则谓词可以上提到WHERE子句中，如查询Q6所示：

Q5:

```
SELECT date_part( 'year',CreateTime) as year, sum(amount)
FROM Orders
GROUP BY Year
HAVING date_part( 'year',CreateTime) = 2022
```

Q6:

```
SELECT date_part( 'year',CreateTime) as year, sum(amount)
FROM Orders
WHERE date_part( 'year',CreateTime) = 2022
GROUP BY Year
```

## 8.2.5 聚集运算的相关规则

- 规则19：聚集与连接运算

- 聚集运算能较大程度减小关系的大小，若将其下移，先进行分组，然后再进行连接操作，连接的效率可以得到很大提高。

- 例如Q7:

```
SELECT C.CID, C.CName, SUM(amount)
```

```
FROM Orders O, Customers C
```

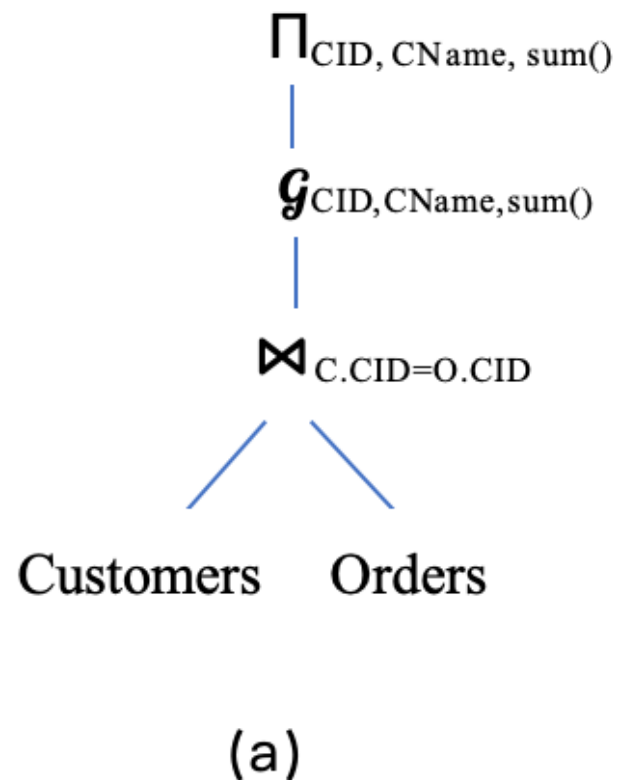
```
WHERE O.CID = C.CID
```

```
GROUP BY C.CID, C.CName;
```

## 8.2.5 聚集运算的相关规则

- Q7两类执行方式:

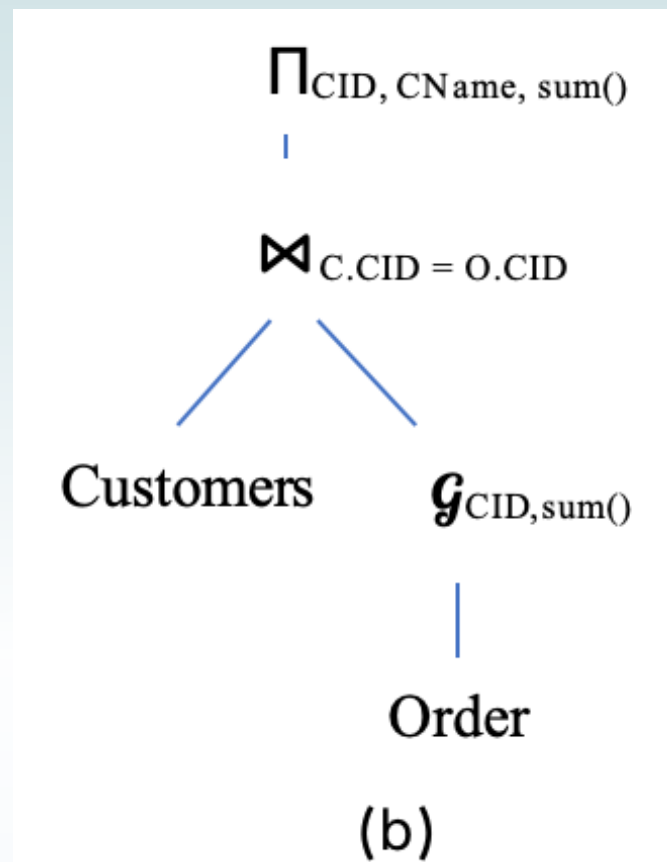
(1)首先将Orders表和Customers表进行连接，得出每个顾客的订单信息，然后按照顾客号对连接结果进行分组，计算出每个顾客的订单总额。



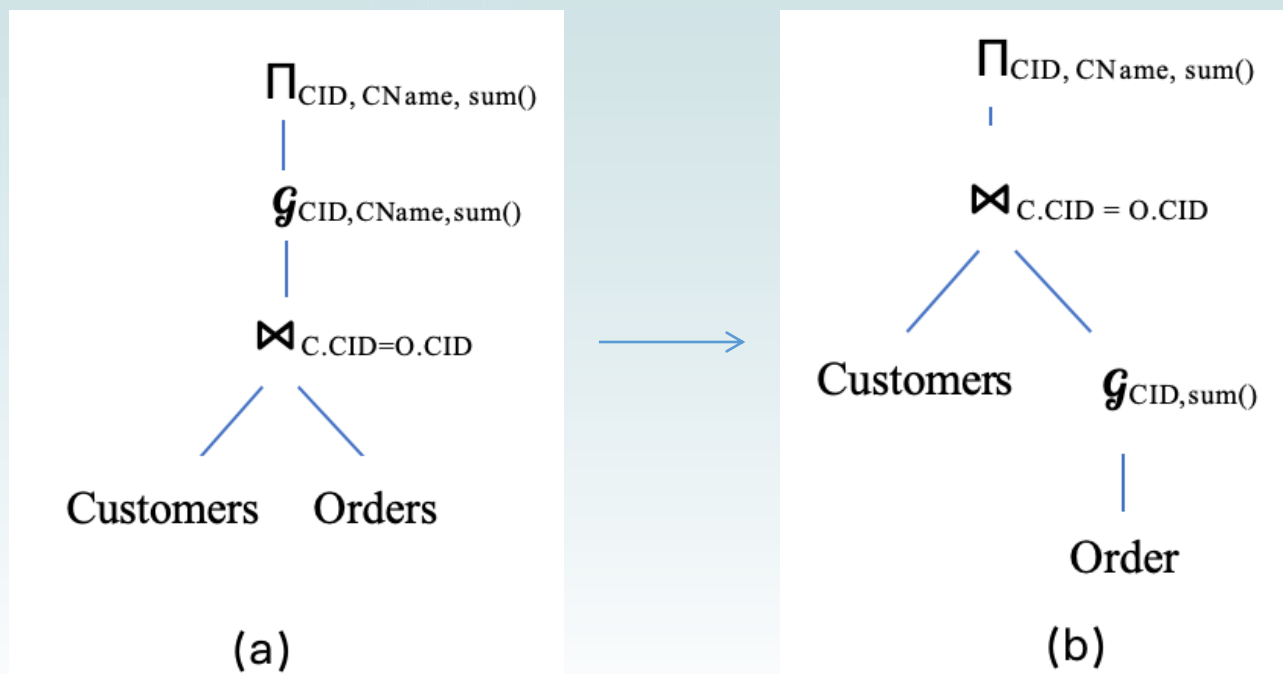
## 8.2.5 聚集运算的相关规则

- Q7两类执行方式:

(2)首先对Orders表按照顾客号进行分组，计算每个顾客号对应的顾客的订单总额，然后，将分组和聚集结果与Customers表进行连接，得到顾客的名字信息。



## 8.2.5 聚集运算的相关规则



- 注：这种变换并不保证一定得到更优的计划，例如，只需要查询部分顾客（假设某个城市的顾客）的订单总额，Orders表和Customers表的连接操作可以过滤掉很多订单元组，则推迟聚集操作的执行可能更加高效。
- 因此，是否需要进行聚集运算的下推需要交给基于代价的优化器去选择。

## 8.2.5 聚集运算的相关规则

- 能够进行聚集运算下推的查询的讨论：
  - 设 $E_j$ 和 $E_k$ 是关系代数表达式， $\theta$ 是连接运算的谓词， $E_j$ 和 $E_k$ 进行连接运算，然后进行聚集运算，记作 $\mathcal{G}_{AeF}(E_j \bowtie_{\theta} E_k)$ ，其中 $\theta$ 中涉及的列称为连接列， $A$ 中的列称为分组列，聚集函数 $F$ 中涉及的列称为聚集列。

## 8.2.5 聚集运算的相关规则

- 若查询 $\mathcal{G}_{AeF}(E_j \bowtie_{\theta} E_k)$  满足下面的条件，则称其在 $E_j$ 上具有不变分组(Invariant Grouping)特性：
  - (1) 聚集列是 $E_j$ 中定义的列，并且不是连接列和分组列。
  - (2)  $E_j$ 中的连接列必须是分组列。
  - (3)  $E_k$ 上有主码，连接谓词是 $E_j$ 中连接列与 $E_k$ 的主码之间的等值连接。

## 8.2.5 聚集运算的相关规则

- 若查询  $\zeta_{AeF}(E_j \bowtie_{\theta} E_k)$  在  $E_j$  上具有不变分组(Invariant Grouping) 特性,则:

$$\zeta_{AeF}(E_j \bowtie_{\theta} E_k) = \zeta_{A'eF}(E_j \bowtie_{\theta} E_k)$$

其中  $A^Z$  是  $A$  中的分组列去掉  $E_k$  中的列。

条件 (1) 确保了聚集运算可以只在上进行计算, 条件 (2) 和 (3) 说明连接列都是分组列, 连接操作不会增加每组元组的个数

## 8.2.6 集合运算的相关规则

- 规则20: 集合并、交的交换律

设 $E_j$ 、 $E_k$ 是关系代数表达式, 则:

$$E_j \cup E_k = E_k \cup E_j$$

$$E_j \cap E_k = E_k \cap E_j$$

## 8.2.6 集合运算的相关规则

- 规则21：集合并、交的结合律

设 $E_j$ 、 $E_k$ 、 $E_l$ 是关系代数表达式，则：

$$(E_j \cup E_k) \cup E_l = E_j \cup (E_k \cup E_l)$$

$$(E_j \cap E_k) \cap E_l = E_j \cap (E_k \cap E_l)$$

# 第八章 查询优化

8.1 查询优化概述

8.2 关系代数表达式等价变换规则

8.3 统计信息

8.4 基数估算

8.5 多表连接的优化

8.6 小结

## 8.3 统计信息

- 现代关系数据库系统普遍采用基于代价的查询优化技术，即：根据对可能的执行计划的代价估算，选择高效的低代价执行计划。
- 在代价估算中，优化器根据系统处理能力、对象大小以及需要读取的数据量等信息来估算出代价，而这些信息主要是系统收集统计数据以及对中间结果集的估算。

## 8.3 统计信息

- 统计信息分类
  - 系统统计信息

系统处理能力是影响执行计划中操作代价的重要因素，系统统计信息主要包括：CPU转速、单数据块的I/O时间、多数据块读的I/O时间、多数数据块平均每次读取的数据块的数量等。

## 8.3 统计信息

- 对象统计信息

(1) 表级的统计数据：元组的数量、表占用的页面数等，这些数据决定了表扫描、连接的代价和内存需求。

(2) 列级的统计数据：要能够推测出列中数据的分布，可以根据条件估算出选择率，例如该列的值域、不同值的个数、空值的比例等。

(3) 多列统计数据：列之间的关系，即相关度，对于多个列条件，可以估算出选择率。

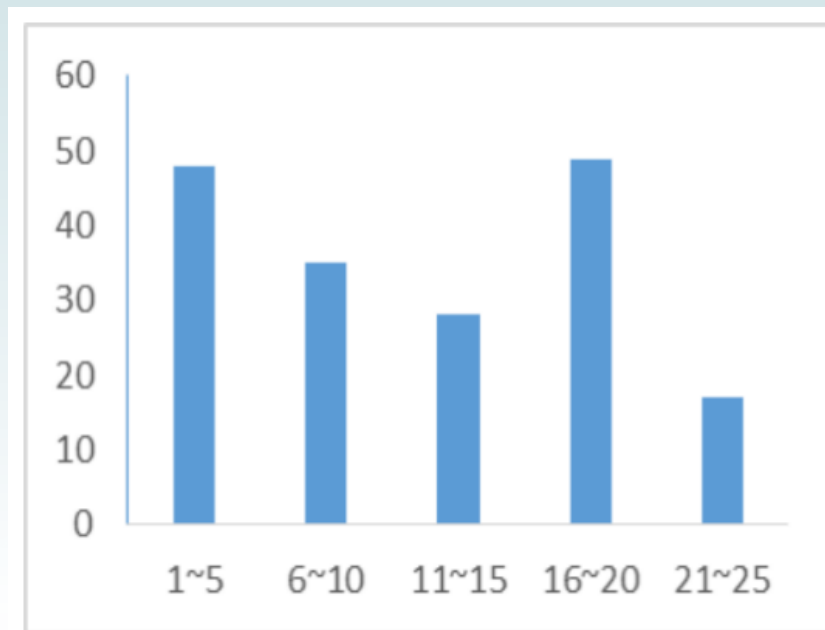
## 8.3 统计信息

- 数据分布表示方法——直方图

直方图是被广泛应用的一种统计图表，它把一个列的取值分成 $k$ 个区间， $k$ 的大小决定了直方图描述该列数据值分布的精确性和内存的使用量。

## 8.3 统计信息

- 根据统计方法，将直方图分类如下：
  - (1) 等宽直方图 (equi-width histogram)：将值的范围划分成值域相等的区间，即对于给定的宽度 $w$ ，对于一组数据，假设最小值为 $v_i$ ，则统计区间 $[v_i, v_i + w)$ 、 $[v_i + w, v_i + 2w)$ .....中元组的个数。



等宽直方图

## 8.3 统计信息

(2) 等频直方图 (equi-depth histogram) :

也叫等高直方图，给定一个频度，对于一组数据，假设最小值为 $\nu_i$ ，统计列出一组数据的序列 $(\nu_i, \nu_j, \nu_k, \dots, \nu_k)$ 使得在每个区间 $[\nu_i, \nu_j)$ 、 $[\nu_j, \nu_k)$ 、 $\dots$ 、 $[\nu_{k-j}, \nu_k)$ 中数据元素的个数占总数据量的比例相等。

- 等频直方图仅需存储区间划分的边界，而不需要存储值的数量，因此其提供了更好的估算信息，并占用更少的空间。

## 8.3 统计信息

- 无论是等宽直方图还是等频直方图，对于每个桶中的值的分布通常假设是均匀的，而为了更准确地估算查询的选择率，还需统计如下信息：
  - 最频值(Most Common Value, MCV): 出现频率最高的前N个数值
  - 以及每个数值出现的频率(Most Common Value frequency, MCVf)

## 8.3 统计信息

- 数据库系统的统计信息通常保存在系统表中，由系统自动或手动进行更新。
  - 表的页面数/表中元组数：通过一次表扫描得到；
  - 直方图：需排序操作。
- 优化器一般不需要准确的统计信息，百分之几的误差通常是可以接受的，因此统计信息的采集可以采用抽样的方法。

# 第八章 查询优化

8.1 查询优化概述

8.2 关系代数表达式等价变换规则

8.3 统计信息

**8.4 基数估算**

8.5 多表连接的优化

8.6 小结

# 第八章 查询优化

**8.4.1 选择运算结果集的估算**

**8.4.2 连接运算结果集的估算**

**8.4.3 其他运算结果集的估算**

## 8.4 基数估算

- 基数估算（Cardinality Estimation, CE）

对中间结果集的规模进行估算，通常使用系统的统计信息，对数据分布假设等来估计一个操作符运算后的结果集大小。统计信息采用下面的符号表示：

- $B(R)$ : 关系 $R$ 的数据块数
- $T(R)$ : 关系 $R$ 的元组数
- $V(R,a)$ : 关系 $R$ 中字段 $a$ 上所具有的不同值的个数
- $S(R)$ : 关系 $R$ 中元组的宽度

## 8.4.1 选择运算结果集的估算

- 选择运算结果集大小的估算依赖于选择谓词。
- 选择率：选择谓词过滤出的数据占总数据量的比例。

## 8.4.1 选择运算结果集的估算

- 等值谓词的选择率估算

- 设  $S = \sigma_{A=c}(R)$ ，其中A是R的属性，c是一个常量。分以下几种情况讨论：

- （1）若系统的统计数据中有A的MCV信息，并且c是其中的一个取值，则直接使用该值的MCVf作为选择率，即  $T(S) = T(R) * MCVf(c)$ 。

- （2）若属性A上有直方图，直方图中桶的个数为K，则可以找出包含值c的桶，假设该桶中的数据值个数（等宽直方图）或桶的宽度（等频直方图）是N，并且取值是均匀分布的，则  $T(S) = T(R) / (K * N)$ 。

## 8.4.1 选择运算结果集的估算

- 等值谓词的选择率估算（续）

- 设  $S = \sigma_{A=c}(R)$ ，其中A是R的属性，c是一个常量。分以下几种情况讨论：

- （3）若没有直方图，则假设属性A中的取值是均匀分布的，关系R中属性A上所具有的不同值的个数为 $V(R,A)$ ，则 $T(S) = T(R) / V(R,A)$ 。

## 8.4.1 选择运算结果集的估算

- 不等值谓词的选择率估算
  - 设  $S = \sigma_{A \neq c}(R)$ ，使用  $\sigma_{A=c}(R)$  的选择率计算。
  - 例如，若  $\sigma_{A=c}(R)$  的选择率为  $1/V(R,A)$ ，则  $T(S) = T(R)(1 - 1/V(R,A))$ 。

## 8.4.1 选择运算结果集的估算

- 范围选择谓词的选择率估算

- 设  $S = \sigma_{A \leq c}(R)$ ，其中A是R的属性，c是一个常量,若属性A上的统计信息有最大值Max(R,A)、最小值Min(R,A)，可假设属性A中的值是均匀分布的，则T(S)估算如下：

(1) 如果  $c < \text{Min}(R,A)$ ，则  $T(S) = 0$ 。

(2) 如果  $c \geq \text{Max}(R,A)$ ，则  $T(S) = T(R)$ 。

(3) 否则，
$$T(S) = \frac{c - \text{Min}(R,A)}{\text{Max}(R,A) - \text{Min}(R,A)} * \frac{1}{V(R,A)} * T(R)$$

## 8.4.1 选择运算结果集的估算

- 例:假设 $\text{Max}(R,A) = 20$ ,  $\text{Min}(R,A) = 0$ ,  $V(R,A)=10$ ,  $S = \sigma_{A \leq j_n}(R)$ , 则

$$T(S) = \frac{c - \text{Min}(R, A)}{\text{Max}(R, A) - \text{Min}(R, A)} * \frac{1}{V(R, A)} * T(R)$$

$$= \frac{j_n - i}{k_i - i} * \frac{j}{j_i} * T(R) = \frac{l}{m_i} * T(R)$$

# 8.4.1 选择运算结果集的估算

- 例(属性A上有更多统计信息)：估计 $col < 1000$ 的选择率，
  - 可以获得的col列上的统计信息如下表所示：
  - 假设在每个直方图的区间中，数据是均匀分布的

统计类别	统计值
直方图	{0,993,1997,3050,4040,5036,5957,7057,8029,9016,9995}
MCV	{12,9994,123,415,4235,3245,125,6745,212,234}
MCVf	{0.00333333,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003}
null值占比	0.1

## 8.4.1 选择运算结果集的估算

统计类别	统计值
直方图	{0,993,1997,3050,4040,5036,5957,7057,8029,9016,9995}
MCV	{12,9994,123,415,4235,3245,125,6745,212,234}
MCVf	{0.00333333,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003}
null值占比	0.1

- 由统计信息可知区间[0, 993)中的元组都是满足条件的，考察1000所在的区间，通过均匀分布的假设，易计算出1000所在区间中满足条件的元组个数，二者合并即可得到选择率。col < 1000选择率可以如下计算：

$$\text{hist\_selectivity} = (1 + (1000 - \text{seg}[2].\text{min}) / (\text{seg}[2].\text{max} - \text{seg}[2].\text{min})) / \text{num\_segs}$$

$$= (1 + (1000 - 993) / (1997 - 993)) / 10$$

$$= 0.100697$$

根据选择率计算公式： $T(S) = \frac{c - \text{MinReAb}}{\text{MaxReAb} - \text{MinReAb}} * \frac{j}{\text{VaReAb}} * T(R)$

## 8.4.1 选择运算结果集的估算

统计类别	统计值
直方图	{0,993,1997,3050,4040,5036,5957,7057,8029,9016,9995}
MCV	{12,9994,123,415,4235,3245,125,6745,212,234}
MCVf	{0.00333333,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003}
null值占比	0.1

- 考虑MCV和空值的情况：

满足 $col < 1000$ 的MCV值共有6个，它们的MCVf分别是0.00333333, 0.003, 0.003, 0.003,0.003 和 0.003，则

$$\begin{aligned} mcv\_selectivity &= \text{sum}(\text{relevant mvcf}) \\ &= 0.00333333 + 0.003 + 0.003 + 0.003 + 0.003 + 0.003 \\ &= 0.01833333 \end{aligned}$$

## 8.4.1 选择运算结果集的估算

统计类别	统计值
直方图	{0,993,1997,3050,4040,5036,5957,7057,8029,9016,9995}
MCV	{12,9994,123,415,4235,3245,125,6745,212,234}
MCVf	{0.00333333,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003,0.003}
null值占比	0.1

- 经过MCV修正后的选择率如下：

$$\begin{aligned}\text{selectivity} &= \text{mcv\_selectivity} + \text{hist\_selectivity} * (1 - \text{mcv\_fraction}) \\ &= 0.01833333 + 0.100697 * 0.96966667 = 0.1159759\end{aligned}$$

- 空值对选择率的影响：

$$\begin{aligned}\text{selectivity} &= \text{mcv\_selectivity} + \text{hist\_selectivity} * (1 - \text{mcv\_fraction} - \text{null\_frac}) \\ &= 0.01833333 + 0.100697 * 0.86966667 = 0.10590615\end{aligned}$$

## 8.4.1 选择运算结果集的估算

- 范围选择运算的选择率计算中，直方图越大，每个区间越小，区间内的数据分布约接近于均匀分布。
- 当MCV越大，MCV对不均匀分布情况下修正的效果就越好。
- 对于其他类型的范围选择运算，例如 $10 < \text{col} < 1000$ ，或者 $\text{col} > 1000$ 的选择率估计，均可转化为“<”类型选择率计算。
- 当范围选择运算的两个操作数均为变量的时候，数据库系统会使用系统给出的一个默认值，例如将选择率指定为 $1/3$ 。

## 8.4.1 选择运算结果集的估算

- 选择条件由多个谓词组成的选择率估算
  - 概率计算

假设A和B是两个相互独立的条件， $P(A)$ 和 $P(B)$ 分别是条件A 和B为 true的概率，使用概率公式计算：

$$P(A \text{ and } B) = P(A) * P(B)$$

$$\begin{aligned} P(A \text{ or } B) &= 1 - (1 - P(A)) * (1 - P(B)) \\ &= 1 - (1 - P(A) - P(B) + P(A) * P(B)) \\ &= P(A) + P(B) - P(A) * P(B) \end{aligned}$$

$$P(\text{not } A) = 1 - P(A)$$

## 8.4.1 选择运算结果集的估算

- 选择条件由多个谓词组成的选择率估算
  - 对于多个选择谓词构成的合取范式，设  $S = \sigma_{F_1 \wedge F_2 \wedge \dots \wedge F_n}(R)$ ，其中R是关系代数表达式， $F_i(i=1,2, \dots, n)$ 是选择谓词。每个 $F_i$ 可以按照前述方法估算其选择率为 $f_i$ ，假设各个选择谓词是相互独立的，则 $\sigma_{F_1 \wedge F_2 \wedge \dots \wedge F_n}(R)$ 的选择率可以估算为：

$$\begin{aligned} f(S) &= f_1 * f_2 * \dots * f_n \\ T(S) &= T(R) * f_1 * f_2 * \dots * f_n \end{aligned}$$

## 8.4.1 选择运算结果集的估算

- 例：R(a,b,c)是一个关系， $S = \sigma_{a=10 \wedge b < 20}(R)$ ， $V(R,a) = 50$ ， $T(R)=3000$ ，如何估算 $T(S)$ ？

$$f(\sigma_{a=10}(R)) = 1/V(R, a) = 1/50$$

$$f(\sigma_{b < 20}(R)) = 1/3$$

$$T(S) = 1/50 * 1/3 * T(R) = 20$$

## 8.4.1 选择运算结果集的估算

- 选择条件由多个谓词组成的选择率估算
  - 对于多个选择谓词构成的析取范式，设  $S = \sigma_{F_1 \vee F_2 \vee \dots \vee F_n}(R)$ ，其中R是关系代数表达式， $F_i(i=1,2,\dots,n)$ 是选择谓词。每个 $F_i$ 可以按照前述方法估算其选择率为 $f_i$ ，假设各个选择谓词是相互独立的，则 $\sigma_{F_1 \vee F_2 \vee \dots \vee F_n}(R)$ 的选择率可以估算为：

$$f(S) = 1 - (1 - f_1) * (1 - f_2) * \dots * (1 - f_n)$$

$$T(S) = T(R) * f(S)$$

- 注：  $f_i = f(\sigma_{F_i}(R))$

## 8.4.1 选择运算结果集的估算

- 部分函数依赖关系：在实际的系统数据库中，某些列之间也可能存在部分相关性
  - 当前很多数据库系统都提供创建多列统计信息的功能来跟踪函数依赖，从而提高估算包含多列的谓词选择率的准确性。

## 8.4.1 选择运算结果集的估算

- 例：  $R(a,b)$  是一个关系,  $T(R)=10000$ ,  $T(\sigma_{a=1}(R)) = 100$ ,  $T(\sigma_{b=1}(R)) = 100$ , 属性  $a,b$  是有函数依赖关系, 每一行中  $a=b$ , 设  $S = \sigma_{a=1 \wedge b=1}(R)$ 。

- 采用不同的方法估算  $T(R)$ :

$$f(\sigma_{a=1}(R)) = 100/10000 = 0.01$$

$$f(\sigma_{b=1}(R)) = 100/10000 = 0.01$$

- 采用计算概率的方法:

$$T(S) = 0.01 * 0.01 * 10000 = 1$$

## 8.4.2 连接运算结果集的估算

- 自然连接的结果集估算

设 $R(X,Y)$ 和 $S(Y,Z)$ 是关系代数表达式，自然连接 $U = R(X,Y) \bowtie S(Y,Z)$ ，其中 $Y$ 是单个属性， $X$ 、 $Z$ 可代表任何属性集。结果集 $U$ 中元组个数跟关系 $R$ 和 $S$ 在连接属性 $Y$ 的取值有直接关系，特别地：

(1)  $R$ 中的 $Y$ 值和 $S$ 中的 $Y$ 值完全没有交集,则:

$$R(X,Y) \bowtie S(Y,Z) = \emptyset, T(R \bowtie S) = T(R)$$

(2)  $R$ 中的 $Y$ 值和 $S$ 中的 $Y$ 值是主外码关系，假设 $Y$ 是 $S$ 的主码，则对于 $R$ 中的每个元组最多与 $S$ 中的一个元组连接， $T(R \bowtie S) = T(R)$ 。

## 8.4.2 连接运算结果集的估算

- 自然连接的结果集估算

一般地，R中的Y值和S中的Y值关系未知，通常做两个假设：

(1) 值集包含：若Y是出现在多个关系中的一个属性，如果 $V(R, Y) \leq V(S, Y)$ ，则R中的每个Y值将是S的一个Y值。

例如：假设R在属性Y上的取值是{1, 3, 6, 9, 10}，如果S在属性Y上的取值个数小于5，则S在属性Y上的取值只能是{1, 3, 6, 9, 10}中的数值。

## 8.4.2 连接运算结果集的估算

- 自然连接的结果集估算

一般的， $R$ 中的 $Y$ 值和 $S$ 中的 $Y$ 值关系未知，通常做两个假设：

(2) 值集保持：如果 $A$ 是 $R$ 的一个属性，但不是 $S$ 的属性，则 $V(R, Y) = V(S, Y)$ 。  
。即当关系 $R$ 与其它关系进行连接操作时，不是连接属性的列取值集合在结果集的该值取值集合保持不变。

## 8.4.2 连接运算结果集的估算

- 自然连接的结果集估算

基于上述两个假设，对 $U = R(X, Y) \bowtie S(Y, Z)$ 结果集规模进行估算：

- 若 $V(R, Y) \geq V(S, Y)$ ，根据值集包含假设，s中的Y值肯定出现在R的Y值集合中，因此可以转换为估算 $\sigma_{R.y=S.y}(R)$ ；
- 等值谓词的选择率为 $1/V(R, Y)$   
→S中的每个元组跟R进行连接运算时，可以生成 $T(R) / V(R, Y)$ 个元组

## 8.4.2 连接运算结果集的估算

- 自然连接的结果集估算（续）

由此，若  $V(R, Y) \geq V(S, Y)$ ，可得：

$$T(R \bowtie S) = T(S) * (T(R)/V(R, Y))$$

同理，若  $V(R, Y) \leq V(S, Y)$ ，可得：

$$T(R \bowtie S) = T(S) * (T(R)/V(S, Y))$$

即：

$$T(R \bowtie S) = (T(S) * T(R)) / \text{Max}\{V(S, Y), V(R, Y)\}$$

## 8.4.2 连接运算结果集的估算

- 例：

有三个关系的统计信息如下，估算 $U = R \bowtie S \bowtie W$ 的结果集大小。

R(a,b)	S(b,c)	W(c,d)
<b>T(R) = 1000</b> <b>V(R,b) = 20</b>	<b>T(S) = 2000</b> <b>V(S,b) = 50</b> <b>V(S,c) = 100</b>	<b>T(W) = 5000</b> <b>V(W,c) = 500</b>

法一：先进行 $U_j = R \bowtie S$ ，再连接 $W$

$$T(U_j) = (T(S) * T(R)) / \text{Max}(V(R, b), V(S, b)) = 40000$$

$$V(U_j, c) = 100 \quad /* \text{根据值集保持假设} */$$

$$T(U) = T(U_j) * (T(W)) / \text{Max}(V(U_j, c), V(W, c)) = 400000$$

## 8.4.2 连接运算结果集的估算

- 例：

有三个关系的统计信息如下，估算 $U = R \bowtie S \bowtie W$ 的结果集大小。

R(a,b)	S(b,c)	W(c,d)
<b>T(R) = 1000</b> <b>V(R,b) = 20</b>	<b>T(S) = 2000</b> <b>V(S,b) = 50</b> <b>V(S,c) = 100</b>	<b>T(W) = 5000</b> <b>V(W,c) = 500</b>

法二：先进行 $U_j = S \bowtie W$ ，再连接R

$$T(U_j) = (T(S) * T(W)) / \text{Max}(V(S, c), V(W, c)) = 20000$$

$$V(U_j, b) = 50 \quad /* \text{根据值集保持假设} */$$

$$T(U) = T(U_j) * T(W) / \text{Max}(V(U_j, c), V(W, c)) = 400000$$

## 8.4.2 连接运算结果集的估算

- 多连接属性自然连接的结果集估算

设 $R(X,Y)$ 和 $S(Y,Z)$ 是关系代数表达式，自然连接 $U = R(X,Y) \bowtie S(Y,Z)$ ，其中 $Y$ 由多个属性组成  $y_j, y_k, \dots, y_n$ ， $X$ 、 $Z$ 可代表任何属性集,假设连接条件独立，则：

$$T(R \bowtie S) = \frac{T(R) * T(S)}{\prod_{i=1}^N \text{Max}\{V(R, y_i), V * S, y_i)\}}$$

## 8.4.2 连接运算结果集的估算

- 多连接属性自然连接的结果集估算

设 $R(X,Y)$ 和 $S(Y,Z)$ 是关系代数表达式，自然连接 $U = R(X,Y) \bowtie S(Y,Z)$ ，其中 $Y$ 由多个属性组成  $y_j, y_k, \dots, y_n$ ， $X$ 、 $Z$ 可代表任何属性集,假设连接条件独立，则：

$$T(R \bowtie S) = \frac{T(R) * T(S)}{\prod_{i=1}^N \text{Max}\{V(R, y_i), V(S, y_i)\}}$$

## 8.4.2 连接运算结果集的估算

- 连接条件独立的假设条件并不总是成立，当连接条件有较强的相关性时，选择率计算会偏小。
- 当要连接的两个关系有参照关系，并且连接条件恰是主码和外码连接时，连接结果集的大小就是外表的大小。
- 因此，通过主外码关系可以对连接运算的结果集大小估算进行一定程度的修正。

## 8.4.2 连接运算结果集的估算

- 其他形式连接运算的结果集估算

(1)  $R \times S$ :  $T(R \times S) = T(R) * T(S)$

(2) 若 $\theta$ 是等值连接，则按照自然连接的方式估算；

(3) 对于其他的 $\theta$ 连接，则可以看作先进行笛卡尔积运算再进行选择运算。

## 8.4.3 其他运算结果集的估算

- 去重运算

设 $R(a_j, a_k, \dots, a_n)$ 是一个关系，去重运算后考虑两个极端情况：

(1)  $R$ 中无重复元组： $T(\delta(R)) = T(R)$

(2)  $R$ 中的元组全部相同， $T(\delta(R)) = 1$

特别地，若可以获得 $R$ 中每个列的不同值个数 $V(R, a_i)$ ，则有：

$$T(\delta(R)) = \text{Min}\{T(R)/2, \prod_{i=1}^n V(R, a_i)\}$$

## 8.4.3 其他运算结果集的估算

- 分组聚集运算

设  $R(a_j, a_k, \dots, a_n)$  是一个关系:

(1) 若已知分组属性A的不同取值个数:  $T(\mathcal{G}_{A,F}(R)) = V(R, A)$

(2) 否则:

$$T(\mathcal{G}_{AeF}(R)) = \text{Min}\{T(R)/2, \bigvee_{i=j}^n V(R, a_i)\}$$

## 8.4.3 其他运算结果集的估算

- 集合运算

若两个输入是同一个关系  $R$ ：将集合操作改写为合取或析取操作，并根据前述方法进行估算，改写如下：

$$\sigma_{F_1}(R) \cup \sigma_{F_2}(R) = \sigma_{F_1 \vee F_2}(R)$$

$$\sigma_{F_1}(R) \cap \sigma_{F_2}(R) = \sigma_{F_1 \wedge F_2}(R)$$

## 8.4.3 其他运算结果集的估算

- 集合运算

若两个输入是不同的关系R、S，则有：

(1) 集合并：

UNION ALL:  $T(R)+T(S)$

UNION: 可能是 $T(R)+T(S)$ ，也可能是 $\text{Max}\{T(R),T(S)\}$ ，建议取 $(T(R)+T(S))/2$

## 8.4.3 其他运算结果集的估算

- 集合运算

(2) 集合交：取值区间 $[0, \text{Min}\{T(R), T(S)\}]$ ，建议取 $\text{Min}\{T(R), T(S)\}/2$

(3) 集合差：取值区间 $[T(R)-T(S), T(R)]$ ，建议取 $(T(R)-T(S))/2$

# 第八章 查询优化

8.1 查询优化概述

8.2 关系代数表达式等价变换规则

8.3 统计信息

8.4 基数估算

8.5 多表连接的优化

8.6 小结

# 第八章 查询优化

**8.5.1 多表连接的查询计划树**

**8.5.2 多表连接顺序的搜索空间**

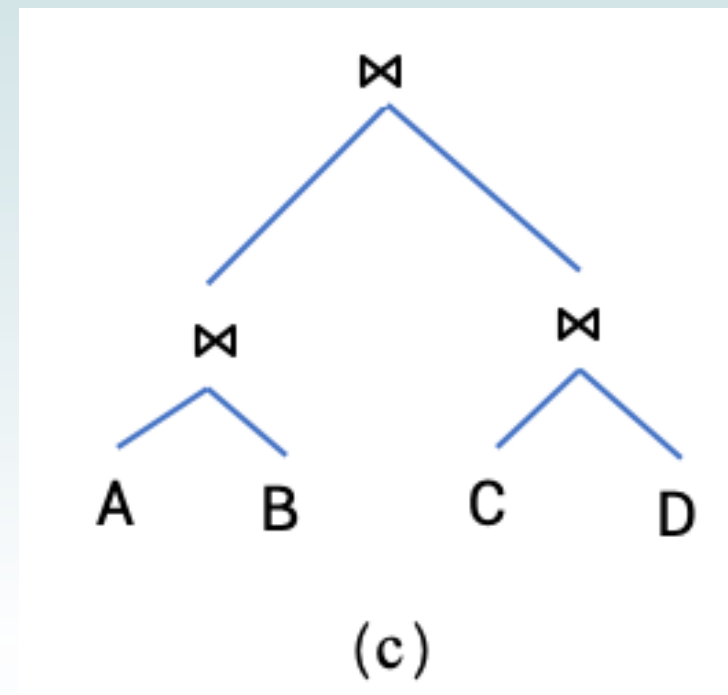
**8.5.3 动态规划**

**8.5.4 贪心算法**



## 8.5.1 多表连接的查询计划树

- 查询计划树的几类基本形态：
  - (3) 稠密树 (bushy tree): 如果一棵连接树既不是左深树也不是右深树, 则称其为稠密树, 如图 (c) 所示。



## 8.5.1 多表连接的查询计划树

- 不同形态的查询计划树的执行方式以及占用的内存数量是不一样的；
- 以哈希连接为例，介绍左深树的执行方式如下：
  - (1) 执行  $A \bowtie B$ ，需要在内存中保留A，并且在计算连接的过程中，需要在内存中保留结果，因此共需要  $B(A) + B(A \bowtie B)$  的内存缓冲区。
  - (2) 继续将  $A \bowtie B$  与  $C$  进行连接，此时，存放A的缓冲区不再需要，可用于存储  $A \bowtie B \bowtie C$  的中间结果。
  - (3) 同理，将该关系与D进行连接时，不再需要保留  $A \bowtie B$  的缓冲区。

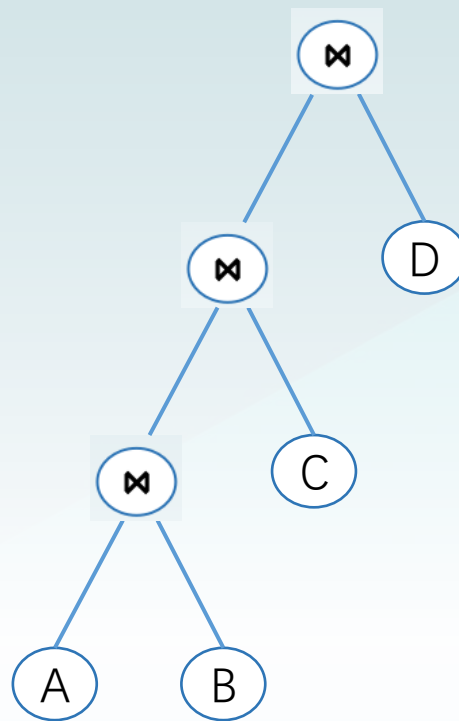
## 8.5.1 多表连接的查询计划树

- 以哈希连接为例，介绍左深树的执行方式如下：

(1) 执行  $A \bowtie B$ ，共需要  $B(A) + B(A \bowtie B)$  的内存缓冲区。

(2) 继续将  $A \bowtie B$  与  $C$  进行连接，此时，存放  $A$  的缓冲区不再需要，可用于存储  $A \bowtie B \bowtie C$  的中间结果。

(3) 同理，将该关系与  $D$  进行连接时，不再需要保留  $A \bowtie B$  的缓冲区。



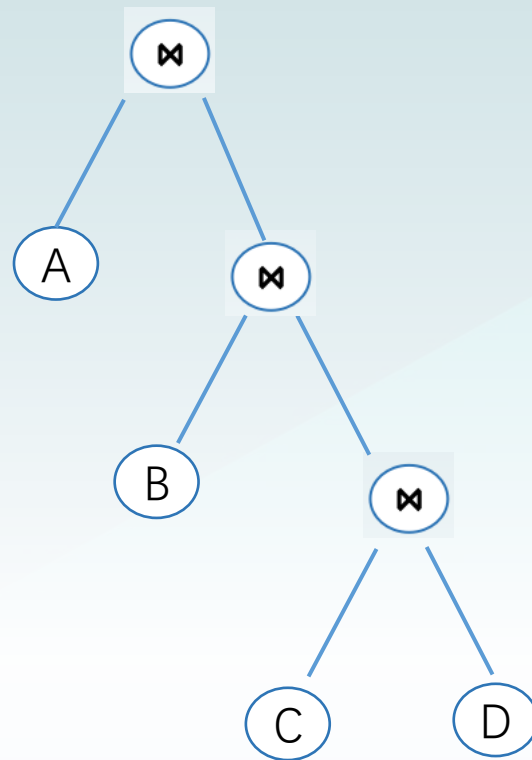
## 8.5.1 多表连接的查询计划树

- 右深树的执行方式如下：

(1) 首先将A读入内存构造哈希表，构建  $B \bowtie C \bowtie D$  用于根连接的探查关系；

(2) 计算  $B \bowtie C \bowtie D$ ，需要将B读入内存缓冲区，计算  $C \bowtie D$  用作B的探查关系；

(3) 计算  $C \bowtie D$  时则需要把C读入内存。因此，一般地，若计算有n个叶子节点的右深树，则需要将n-1个关系同时读入内存。



## 8.5.1 多表连接的查询计划树

- 左深树vs.右深树

上例中，若A很小，我们可以预期  $A \bowtie B$  远小于B以及  $A \bowtie B \bowtie C$  会小于C，在概率上，使用左深树计算连接需要的内存量会小于右深树。

- 因此，有些数据库系统在考虑多个表的连接顺序时，为了减少搜索空间，只考虑左深树，以提高生成执行计划的效率。
- 如果内存足够大，由于右深树中的构建表都在内存中，使得各个连接操作可以并行执行，从而形成流水线来提高查询的执行效率。

## 8.5.2 多表连接顺序的搜索空间

- 问题：给定 $n$ 个关系的集合  $S_n = \{R_j, R_k, \dots, R_n\}$ , 请问这 $n$ 个关系按照什么顺序进行连接执行的总代价最小？
- 法一：穷举法  
缺点：关系的每一种排列方式下不同的执行方式数量规模过大，效率较低。

## 8.5.2 多表连接顺序的搜索空间

- 穷举法的排列数量及不同树形结构数量计算：

假设任意两个关系进行连接的最小代价可以使用函数Mincost(x,y)计算，则有

- (1)  $n$ 个关系的排列方式有 $n!$ 种。
- (2) 对于每种排列方式，可以有 $T(n)$ 种树形结构，计算 $T(n)$ 递归公式如下：

$$T(n) = \begin{cases} 1 & n = 1 \\ \sum_{i=1}^{n-1} T(i) * T(n-i) & n > 1 \end{cases}$$

## 8.5.2 多表连接顺序的搜索空间

$$T(n) = \underbrace{\sum_{i=1}^{n-1}}_1 T(i) * T(n-i) \quad \begin{matrix} n=1 \\ n>1 \end{matrix}$$

该递归公式含义：

- 对于一个n个关系的排列，可以任选一个i（ $1 \leq i < n$ ），i把n个关系分成两个分别有i和n-i个关系的集合；
- 前i个关系有T(i)个树形结构，后n-i个关系有T(n-i)个树形结构，；
- 对于每个i，树形结构的个数是 $T(i) * T(n-i)$ ，因此，T(n)是所有i的树形结构总和。

## 8.5.3 动态规划

- 法二：动态规划
- 最优子结构性质：求 $n$ 个关系进行连接的连接顺序的最优解包含求其子问题的最优解。
- 在动态规划算法中，问题的最优子结构特性使其能够以自底向上的方式递归地从子问题的最优解逐步构造出整个问题的最优解，可以在相对小的子问题空间中考虑问题。

## 8.5.3 动态规划

- 给定n个关系的集合 $S_n = \{R_j, R_k, \dots, R_n\}$ ，将S中所有关系连接 $R_j \bowtie R_k \bowtie \dots \bowtie R_n$ 记作 $Join(S_n)$ ，现在要求求解这n个关系的连接顺序最优解问题。
- 设以最优连接顺序计算 $Join(S_i)$  ( $1 \leq i < n$ ) 的代价为 $JCost(S_i)$ ，利用其最优子结构性质，原问题的以最优连接顺序计算 $Join(S_i)$ 的代价可以递归定义如下：
- $JCost(S_n)$

$$= \begin{cases} 0 & , n = 1 \\ \text{Min}\{JCost(S_i) + JCost(S_n - S_i) + \text{Mincost}(S_i, S_n - S_i), 1 \leq i < n\} & , n > 1 \end{cases}$$

## 8.5.3 动态规划

- 使用递归算法来计算  $JCost(S_n)$ ，将耗费指数级的计算时间。
- 子问题的重叠性质：用递归算法自顶向下计算时，每次产生的子问题并不总是新问题，子问题会重复计算多次。
- 该特性使得对每个子问题只求解一次，从而规避大量的重复计算，获得较高的解题效率。

## 8.5.3 动态规划

- 动态规划主要思想：
  - 利用求解问题最优子结构和子问题重叠性质，在计算过程中，保存已解决的子问题答案，根据需求查找，避免大量的重复计算，从而提高性能。
  - 自底向上计算每层（ $i$ 个关系的连接， $2 \leq i \leq n$ ）子问题的最优解。使用哈希表 `BestJoinOrder` 保存每个子问题的信息，包括参加连接的关系集合（哈希表的 `key`）、最优连接顺序 `joinorder` 和该连接顺序执行代价 `cost`。
  - 在计算第  $i$  层的最优解时，计算其中的每个组合的执行代价，找出最优的执行顺序作为该层的最优解。

## 8.5.3 动态规划

- 动态规划算法的形式化表示:

FOR S中的每个关系R

Join.key = {R}; Join.cost = 0; Join.joinorder = R;

把Join插入到哈希表BestJoinOrder;

END

FOR i=2; i<=n;i++

FOR S中i个关系的每种组合 $S_i$  /\*遍历第i层 $S_i$ 中的关系的所有组合方式 $S_k$ 和 $(S_i-S_k)$ \*/

Mincost =  $\infty$ ;

FOR k=1;k<=i;k++

把 $S_i$ 中的关系分成两个集合 $S_k$ 和 $(S_i-S_k)$

对每一种分法计算 $S_i$ 的最优执行代价:

## 8.5.3 动态规划

- 动态规划算法的形式化表示（续）：

$JCost(S_i) = JCost(S_k) + JCost(S_i - S_k) + Mincost(S_k, S_i - S_k);$

IF  $JCost(S_i) < Mincost$  /\*计算每种组合的代价并与当前最小值比较\*/

$Mincost = JCost(S_i);$

$Join.key = S_i; Join.cost = JCost(S_i);$

$Join.joinorder = S_k \text{ 和 } (S_i - S_k) \text{ 的最优连接顺序};$

ENDIF

//Join中记录了 $S_i$ 的最优执行顺序和执行代价

把Join插入到哈希表BestJoinOrder;

END

END

END

## 8.5.3 动态规划

- 例：以  $R \bowtie S \bowtie T \bowtie U$  为例,假设每个关系的大小为1000个元组，关系的属性以及相应的统计信息如表所示。

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

- 定义两个关系X和Y连接的最小代价Mincost(X,Y)是该连接中间结果关系大小的和，即  $Mincost(X,Y) = TempT(X) + TempT(Y)$

$$TempT(R) = \begin{cases} 0 & R \text{ 为基表} \\ \infty & R \text{ 为中间临时关系} \end{cases}$$

## 8.5.3 动态规划

- 例（续）：对子问题 $JCost(S_i)$ 采用表格记录以下的信息：
  - （1）关系集合 $S_i$ 中所有关系连接后的结果集大小的估计值。
  - （2）该连接执行的代价。
  - （3）该子问题的解：最优连接顺序

## 8.5.3 动态规划

- 例（续）：当 $n=1$ 时， $S_1$ 中只有一个关系，并且是基表，不存在连接顺序的问题，其表格如下所示。

	{R}	{S}	{T}	{U}
大小	600	800	1000	1200
代价	0	0	0	0
最优顺序	R	S	T	U

$n=1$ 的信息

## 8.5.3 动态规划

- 例（续）：当 $n=2$ 时，4个关系有6种组合 $\{R,S\}$ 、 $\{R,T\}$ 、 $\{R,U\}$ 、 $\{S,T\}$ 、 $\{S,U\}$ 和 $\{T,U\}$ ，对每种组合求出其最优顺序和最小连接代价。此时参加连接运算的都是基表，代价为0，结果集大小使用公式估算。

	$\{R,S\}$	$\{R,T\}$	$\{R,U\}$	$\{S,T\}$	$\{S,U\}$	$\{T,U\}$
大小	4000	600000	12000	2000	960000	1000
代价	0	0	0	0	0	0
最优顺序	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$

$n=2$ 的信息

## 8.5.3 动态规划

$$JCost(S_n) = \begin{cases} 0, & n = 1 \\ \min\{JCost(S_i) + JCost(S_n - S_i) + Mincost(S_i, S_n - S_i), 1 \leq i < n\}, & n > 1 \end{cases}$$

- 例（续）：当 $n=3$ 时，4个关系有4种组合 $\{R,S,T\}$ 、 $\{R,S,U\}$ 、 $\{R,T,U\}$ 和 $\{S,T,U\}$ ，对每种组合求出其最优顺序和最小连接代价。以 $\{R,S,T\}$ 为例：

$$\begin{aligned} JCost(\{R, S\}) + JCost(\{T\}) + Mincost(\{R, S\}, \{T\}) &= 0 + 0 + 4000 \\ JCost &= \min \begin{cases} JCost(\{R, T\}) + JCost(\{S\}) + Mincost(\{R, T\}, \{S\}) = 0 + 0 + 6000000 \\ JCost(\{S, T\}) + JCost(\{R\}) + Mincost(\{S, T\}, \{R\}) = 0 + 0 + 2000 \end{cases} \end{aligned}$$

因此，对于 $\{R,S,T\}$ 来说，最小代价是2000，最优顺序是 $(S \bowtie T) \bowtie R$ ，其结果集大小是 $2000 * 1000 / 200 = 10000$ 。

## 8.5.3 动态规划

- 例（续）：当 $n=3$ 时，表格如下图：

	{R,S,T}	{R,S,U}	{R,T,U}	{S,T,U}
大小	10000	80000	10000	2000
代价	2000	4000	1000	1000
最优顺序	$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(T \bowtie U) \bowtie S$

$n=3$ 的信息

## 8.5.3 动态规划

- 例（续）：当 $n=4$ 时，只有一个组合 $\{R, S, T, U\}$ ，子问题有两种：
  - （1）分成三个关系集合和一个关系的集合；
  - （2）分成两个关系集合和两个关系的集合。因此，有以下的连接方式：

$JCost(\{R, S, T, U\}) =$

$$\text{Min} \left\{ \begin{array}{l} JCost(\{R, S, T\}) + JCost(\{U\}) + \text{Mincost}(\{R, S, T\}, \{U\}) = 2000 + 0 + 10000 = 12000 \\ JCost(\{R, T, U\}) + JCost(\{S\}) + \text{Mincost}(\{R, T, U\}, \{S\}) = 1000 + 0 + 10000 = 11000 \\ JCost(\{S, T, U\}) + JCost(\{R\}) + \text{Mincost}(\{S, T, U\}, \{R\}) = 1000 + 0 + 2000 = 3000 \\ JCost(\{R, S, U\}) + JCost(\{T\}) + \text{Mincost}(\{R, S, U\}, \{T\}) = 4000 + 0 + 80000 = 84000 \\ JCost(\{S, T\}) + JCost(\{R, U\}) + \text{Mincost}(\{S, T\}, \{R, U\}) = 0 + 0 + 14000 = 14000 \\ JCost(\{S, U\}) + JCost(\{R, T\}) + \text{Mincost}(\{S, U\}, \{R, T\}) = 0 + 0 + 1560000 = 1560000 \\ JCost(\{T, U\}) + JCost(\{R, S\}) + \text{Mincost}(\{T, U\}, \{R, S\}) = 0 + 0 + 5000 = 5000 \end{array} \right.$$

由上式知最小代价是 $\{S, T, U\}$ 与 $\{R\}$ 的连接，代价为3000，因此最优顺序是  $((T \bowtie U) \bowtie S) \bowtie R$ 。

## 8.5.4 贪心算法

- 动态规划缺点：当参与连接的关系数目较大时，其效率依然会较低。
  - 因此考虑采用启发式的方法寻找最优执行顺序→贪心算法
- 贪心选择：所作的每一个选择都是当前状态下某种意义的最好选择。
- 贪心算法：通过每次所作的贪心选择形成的最终结果时问题的一个最优解。该算法并不能保证最终能产生最优解，但在很多情况下可以产生整体最优解，或最优解的近似解。

## 8.5.4 贪心算法

- 例：以  $R \bowtie S \bowtie T \bowtie U$  为例,解释贪心算法过程：
  - 当  $n=1$  时，与动态规划算法求解一致。
  - 当  $n=2$  时，从下表中可以看出  $T \bowtie U$  的代价最小，选择其为当前最优树。

	{R,S}	{R,T}	{R,U}	{S,T}	{S,U}	{T,U}
大小	4000	600000	12000	2000	960000	1000
代价	0	0	0	0	0	0
最优顺序	$R \bowtie S$	$R \bowtie T$	$R \bowtie U$	$S \bowtie T$	$S \bowtie U$	$T \bowtie U$

$n=2$  的信息

## 8.5.4 贪心算法

- 例：以  $R \bowtie S \bowtie T \bowtie U$  为例,解释贪心算法过程：
  - 当  $n=3$  时，只需要考虑  $(T \bowtie U) \bowtie R$  和  $(T \bowtie U) \bowtie S$  两种情况，下表中可以看出  $(T \bowtie U) \bowtie S$  的代价最小。
  - 当  $n=4$  时，没有可选的，只有  $((T \bowtie U) \bowtie S) \bowtie R$ ，得出问题的最优解。

	$\{R,S,T\}$	$\{R,S,U\}$	$\{R,T,U\}$	$\{S,T,U\}$
大小	10000	80000	10000	2000
代价	2000	4000	1000	1000
最优顺序	$(S \bowtie T) \bowtie R$	$(R \bowtie S) \bowtie U$	$(T \bowtie U) \bowtie R$	$(T \bowtie U) \bowtie S$

$n=3$  的信息

## 8.5.4 贪心算法

- 贪心算法 vs. 动态规划
- 与动态规划算法相比，贪心算法丢失最佳计划的可能性更大，但其计算量明显减少。
- PostgreSQL系统采用动态规划方法来做多表连接的顺序选择。
- MySQL系统采用贪心方法求解该问题。

# 第八章 查询优化

8.1 查询优化概述

8.2 关系代数表达式等价变换规则

8.3 统计信息

8.4 基数估算

8.5 多表连接的优化

8.6 小结

# 小结

- 系统介绍了查询优化的基本概念、方法以及关键问题，即：在优化过程中如何充分利用各种运行数据或历史数据来实施优化。
- 主要介绍了两类优化方式，其中逻辑优化主要指基于启发式规则对关系代数表达式等价变换，其中预估成本需要的运行数据或历史数据都是系统统一维护的统计信息；物理优化则涉及操作符选择、操作顺序、执行方式等。
- 重点讲解了多表连接顺序的动态规划方法和贪心算法，后者虽然不能保证获得最优算法，但是能够降低查询计划的选择代价。

