高级数据库系统及其应用

第2部分　关系数据库系统实现

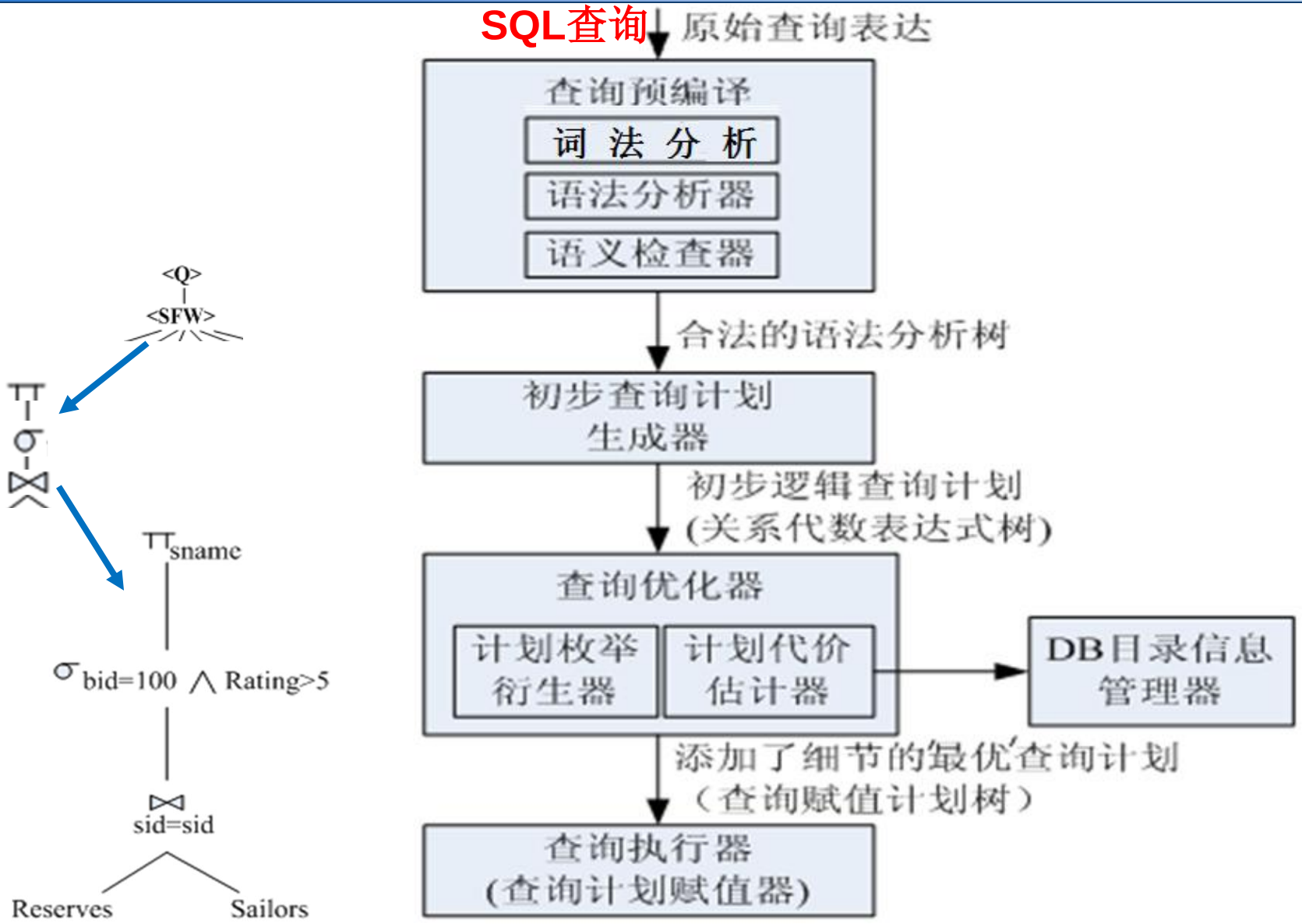
第7章 查询处理与优化

xshxie@ustc.edu.cn

LOGO

第7章 查询处理与优化

7.1 查询处理过程简介


SQL查询
原始查询表达
查询预编译
词法分析
语法分析器
语义检查器
合法的语法分析树
初步查询计划
生成器
初步逻辑查询计划
(关系代数表达式树)
查询优化器
计划枚举
衍生器
计划代价
估计器
DB目录信息
管理器
添加了细节的最优查询计划
(查询赋值计划树)
查询执行器
(查询计划赋值器)
<Q>
<SFW>
sname
bid=100 ∧ Rating>5
sid=sid
Reserves
Sailors

7.1 查询处理简介

- 即使是一个单操作符赋值，也有许多不同赋值方式。
  - 每种方法都可能在某些场合下优于其它方法。
- 一个实际查询，通常是多个操作符组合的代数树。
  - 发现/找到一棵好查询计划树是一项重要的挑战。
- **DBMS**必须考虑可替换的、尽量多的可选计划树，并选择其中估算代价最低者 。

7.2 查询优化综述

7.2.1 查询赋值计划

7.2.2 流水线赋值

7.2.3 IBM System R优化器

基本目标

- 重写初步逻辑查询计划，生成优化的逻辑查询计划。
- 为给定查询找到一个好的查询赋值计划。

基本步骤

- 枚举赋值表达式的各种可选赋值计划；
- 估计每个枚举计划的代价，选出具有最少代价的那个计划。

7.2.1 查询赋值计划

- 查询赋值计划
  - 简称执行计划，或计划
  - 是一棵带有节点标注的扩展关系代数树，节点标注可包括对应关系操作符的存取和实现方法等说明。
- 考虑下面SQL查询例句：

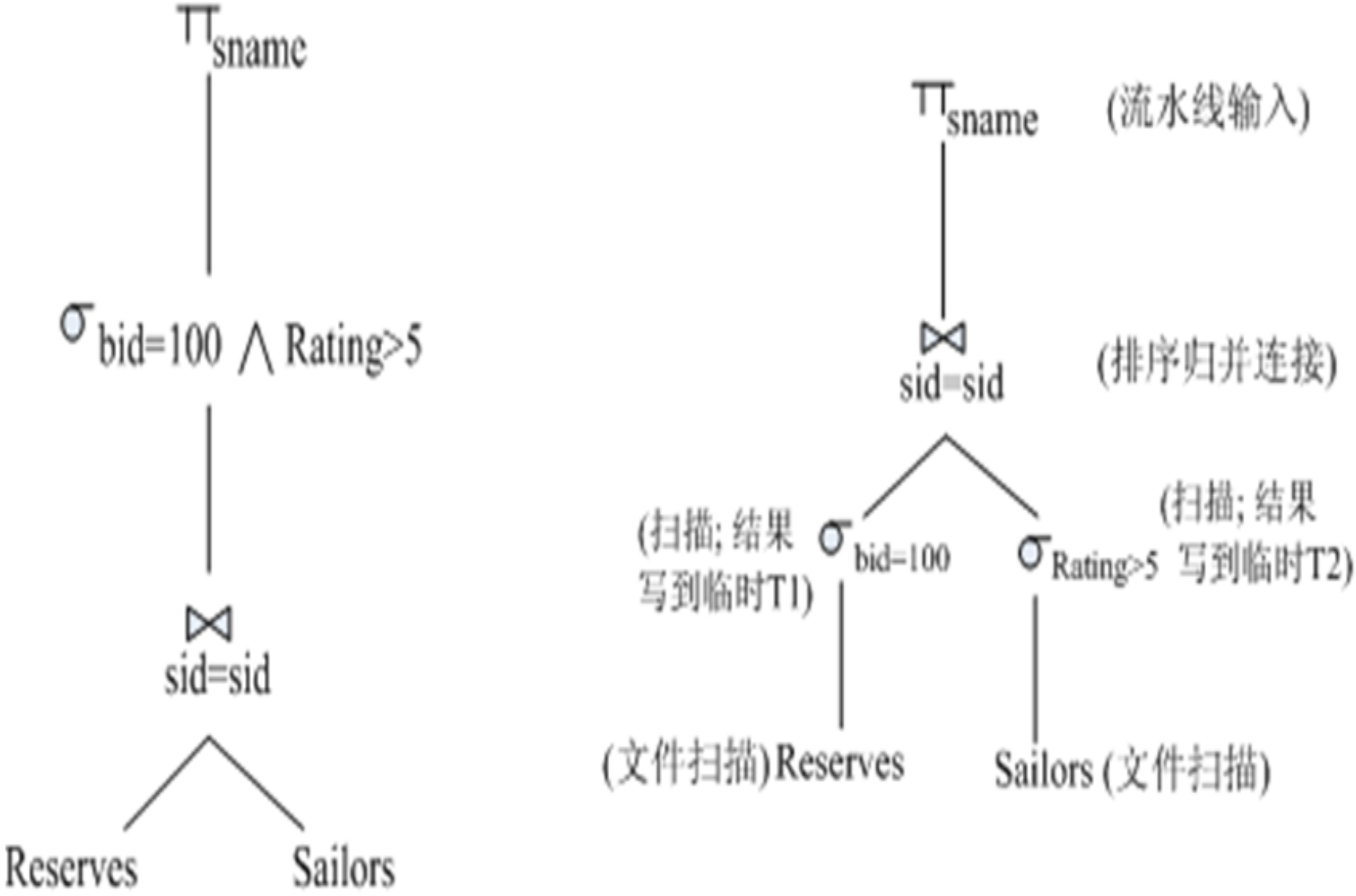
例7.3

```
SELECT S.sname
    FROM    Reserves R, Sailors S
    WHERE  R.sid = S.sid AND R.bid=100
                AND S.rating > 5
```

- 对应的关系代数表达式：

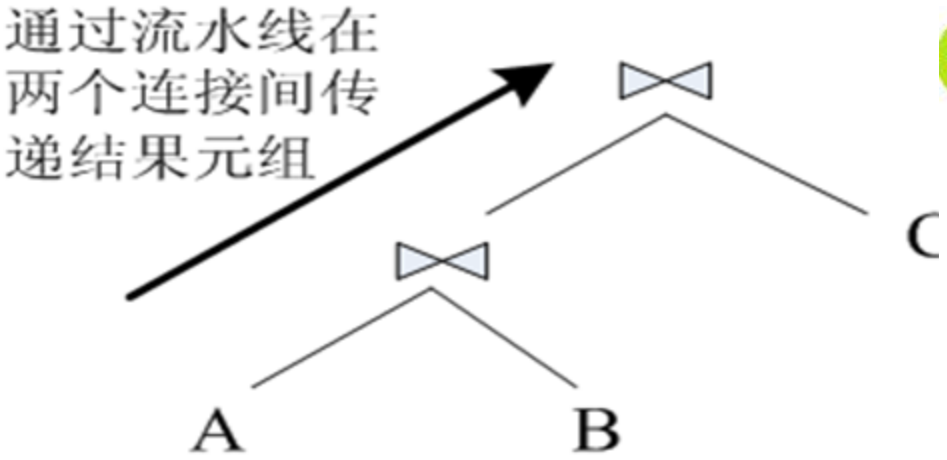
$$\Pi sname(\sigma_{bid=100\ AND\ rating>5}(Reserves \bowtie_{sid=sid} Sailors))$$

图7.3 一个典型的查询赋值计划树



7.2.2 流水线方式赋值

- 流水线方式与物化方式的概念
- 流水线赋值
  - 在处理多关系连接时，通常采用流水线方式在上下层连接操作间传递元组。
  - 流水线方式允许我们避免创建和读取**临时关系**文件，能节省大量的磁盘I/O操作。从而有益于减少查询赋值代价。--但这只是启发式定律



但

有时当物化代价不大且有助于利用某些其它特性时，或当主存严重不足时，采用物化方式反而可能开销更小。

7.2.3 IBM System R优化器

- 对现代**RDBMS**查询优化器具有重要影响，其实现采用了一些重要设计决策，包括：
  - 利用DB实例中的统计信息，来辅助估计查询赋值计划代价；
  - 对二元连接操作符，只考虑内层是基关系(非临时关系)的赋值计划。这个启发式决策，能减少大量需要考虑的可选赋值计划。
  - 优化工作的焦点及其实施，主要集中在无嵌入子查询的SQL查询类。对嵌入查询处理，只是采用通行的方式进行一般性处理。
  - 对投影，不自动进行删除重复记录的操作（除非查询中包含DISTINCT请求）

7.3 关系代数的等价规则

7.3.1 选择

7.3.2 投影

7.3.3 叉积与连接

7.3.4 选择、投影与连接

7.3.5 其它等价规则

7.3 关系代数等价规则

- 代数等价定义
  - 如果两个关系代数表达式，对所有输入关系实例都能产生相同的结果，则它们被认为等价的 .
- 两个与选择操作有关的等价规则
  - 级联选择的等价规则
  - 可交换性规则
- 级联投影规则
- 与叉积、连接有关的等价规则
  - 交换律、结合律

一些同时包含两个以上操作符的等价规则

- 选择、投影和连接
  - $\Pi_a(\sigma_c(R)) \equiv \sigma_c(\Pi_a(R))$
- 也可以合并选择与叉积，来产生一个新连接
  - $R \bowtie_c S \equiv \sigma_c(R \times S)$
- 允许交换选择与叉积，或选择与连接：
  - $\sigma_c(R \times S) \equiv \sigma_c(R) \times S$； $\sigma_c(R \bowtie S) \equiv \sigma_c(R) \bowtie S$
- 若*c1*同时含*R*与*S*的属性;*c2*只含*R*的属性;*c3*只含*S*的属性
  - $\sigma_c(R \times S) \equiv \sigma_{c1 \wedge c2 \wedge c3}(R \times S) \equiv \sigma_{c1}(\sigma_{c2}(R) \times \sigma_{c3}(S))$
- 交换投影与叉积
  - *若a1*是*a*中只出现在*R*的属性子集，*a*2是*a*中只出现在*S*：
  - $\Pi_a(R \times S) \equiv \Pi_{a1}(R) \times \Pi_{a2}(S)$
  - $\Pi a(R \bowtie S) \equiv \Pi a1(R) \bowtie \Pi a2(S)$

7.4 基于等价和启发式规则的查询优化

7.4.1 下推选择与下推投影

7.4.2 利用索引改进计划

前节知识

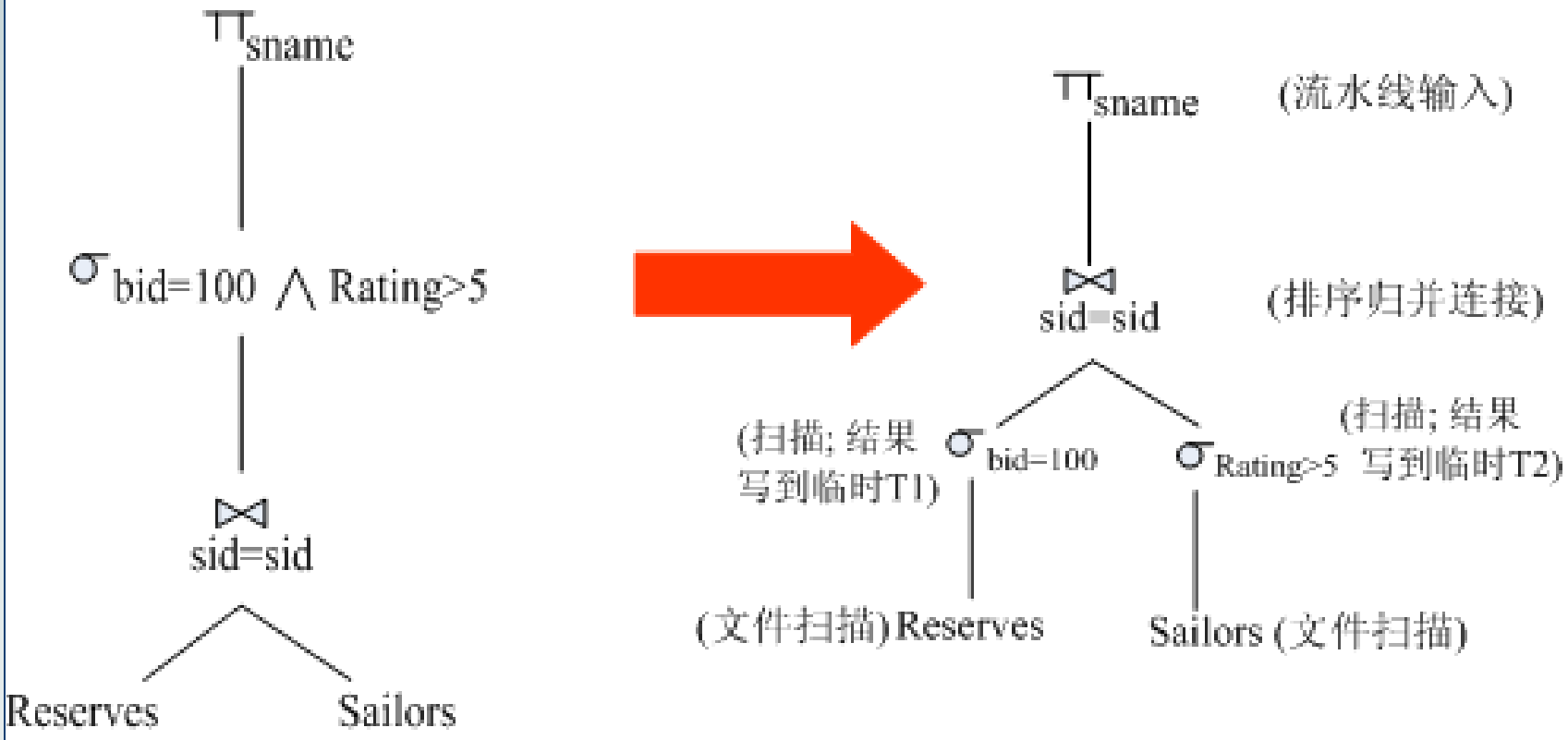
关系代数等价规则允许我们下推选择和投影到连接之前；转换叉积到连接；选择不同的连接顺序等。

本节讨论

如何利用关系代数的等价规则和一些启发式(Heuristic)优化规则，来等价变换查询赋值计划，以获得具有更小代价的查询赋值计划。

下推选择

- 连接是一个相对昂贵的操作。有个很好的启发式规则是：
  - 尽可能下推选择，以减小参与连接的关系大小。



例题 7.4

❖假设

1) 只有 5 个可用的缓存页；
2) 共有100 条船，且船员被均匀分派到各条船上
3) 水手的rating值均匀分布在1～10范围内。

试估计图7.5计划的代价。

(1)执行选择步，代价 ～ **1760**

(2)执行连接步，

- 利用B=5， 完成T1, T2排序连接，～2260， + 1760 = 4020
  - 排序T1: 4 x 10 =40; 排序T2: 2 x 4 x 250 =2000 --4阶段
  - 合并T1,T2, 250 + 10 = 260;
- 若用块嵌套连接，T1作外层关系，对T1的每3个页，扫描一次T2
  - 这一步代价 10 + 4 x 250 = 1010 次 + 1760 = 2770

利用索引改进计划

例题7.5 对例7.3查询,设

1）在Reserves.*bid*上有静态聚集散列索引，
Sailors.*sid*上有静态散列索引；

2）共有*100*条船，且船员被均匀分派到各条船上。

❖ 计算右图所示改进计划

- 利用bid索引执行bid=100,均匀分布，满足条件1000个元组聚集分布在约10个页，即仅10次I/O
- 每得到一个sid,利用sailors.sid索引及相应元组,合计约1.2次
- 总计：10 + 1000x1.2 = 1210

这个计划没有利用sailors.sid的聚集特性 ??

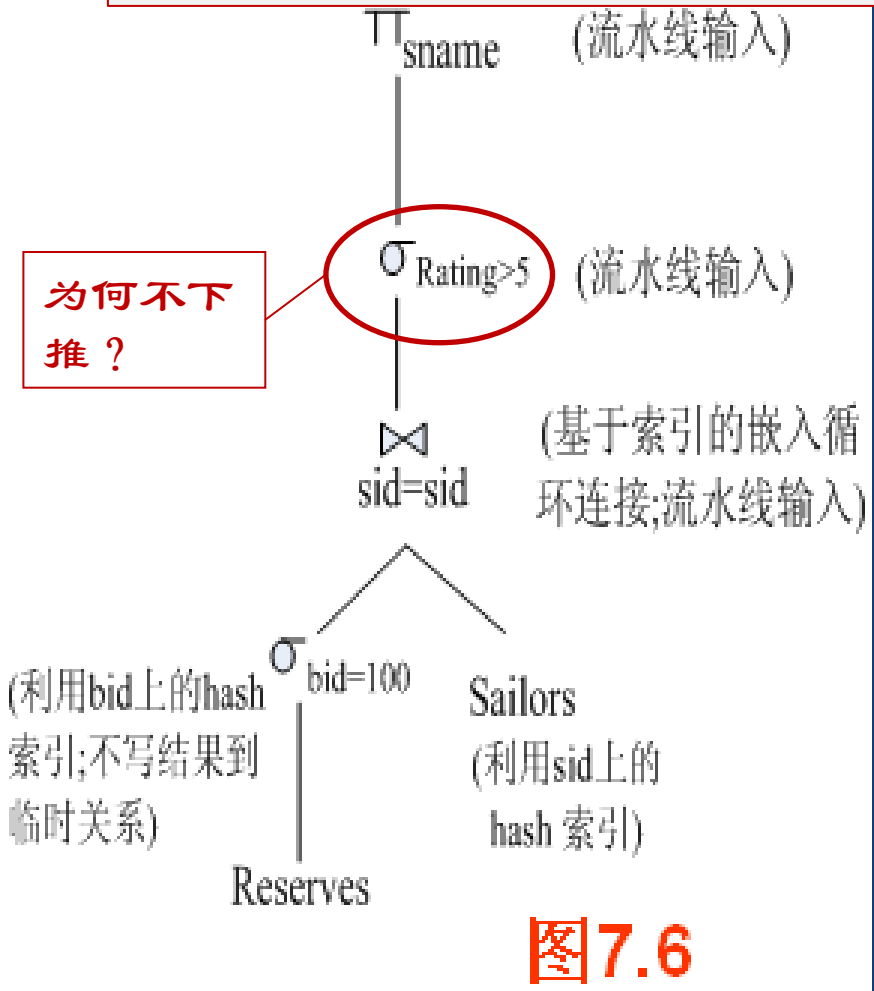


图7.6

小结

- 恰当使用下推选择、下推投影和索引这些启发式优化规则，在大多数情况下会非常有效。
- 特别地，如果外层关系被选择参与连接的元组只需与内层的一个元组连接(常用的外键-主键连接就是这种情况)，则整个连接操作将变为平凡微不足道。
- 流线方式也未必总是最好的方式
  - 有时当物化代价不大且有助于利用某些其它特性时，或当主存严重不足时，采用物化方式可能开销更小。

7.5 作为中间结果的操作符输出大小估计

- 对每个枚举的候选赋值计划，必须估算它的赋值执行代价。
  - 这至少包括以下方面工作：
    - 对计划树中的每个节点，我们必须估计执行相关操作的代价。
    - 是用流水线还是物化方式将输出结果传给父节点，对代价也有重要影响。
    - 对计划树中的每个节点，我们也必须估计结果的大小，并考虑结果是否要求排序问题。
- 估计计划树中间节点(中间关系)的大小，是估算整个赋值执行计划代价的难点。

7.5 作为中间结果的操作符输出大小估计

7.5.1 选择输出的大小估计

7.5.2 连接大小的估计

7.5.3 消除重复操作的大小估计

7.5.4 其它操作符的结果大小估计

- 作为估算计划代价方法,只需要满足以下两条准则
  - 能给出较合理的相对大小估计值。
  - 计算简便、相关处理逻辑一致；
- 按这两个准则，在估计中间关系大小时，有时我们甚至可以只估计临时关系的元组数，不必去计算以字节为单位的绝对结果大小估值。

7.5.1 选择输出的大小估计

Attr =value

$$f_{attr=value}=\begin{cases}1/NKeys(I)\text{或}\\1/V(R,Attr)\text{或}\\1/10\end{cases}$$

Attr1= Attr

$$f_{attr1=attr2}=\begin{cases}1/\max\{NKeys(I_1),NKeys(I_2)\}\text{或}\\1/\max\{V(R,Attr1),V(R,Attr2)\}\text{或}\\1/10\end{cases}$$

Attr ≠value

$$f_{attr\neq value}=1-f_{attr1=value}$$

7.5.1 选择输出的大小估计

Attr > value

- 如果查询关系在属性列$Attr$上有索引，则类条件项的减少因子可用(IHIGH(I)-value)/(IHIGH(I)-Low(I))来近似估算，ILow(I)和IHigh(I)是索引I的最小、最大键值。
- 如果该列不是算术类型，或没有索引可用，则简单采用一个小于1/2的因子(如1/3)。
- 对于范围选择，也可以用类似方法构造大小估计公式。

Attr IN (*list of values*)

- 用 Attr_name = value 类型条件项减少因子乘以值列表中值的个数来估计
- 但最大不超过1/2，这是基于“每个选择将删除至少一半候选元组”这个启发式经验考虑的。

7.5.1 选择输出的大小估计

- 对基本查询块：
  - SELECT *attribute_list*
    FROM *relation_list*
    WHERE *term*1 ∧ *term*2 ∧…∧ *term*n
  - 输出结果的最大元组数目，是FROM语句列表关系叉积所产生的元组总数。
  - WHERE子句中的每个项将删除这些潜在结果元组中的某些元组。
- 可采用如下估算模型：
  - 将每个项视为一个减少因子；同时采用一个非实际的简化假设：每个项对应的条件测试在统计上是相互独立的。
  - 总的减少因子＝各减少因子的乘积

7.5.2 连接大小的估计

- 只考虑自然连接（等值连接、广义θ条件连接总可以转为自然连接）
  - 两关系R和S在字段Y上的自然连接：
    - R(X,Y) ⋈S(Y,Z)
- 三个特殊连接的输出结果大小
  - R与S有不相交的Y值集，则 T(R⋈S)=0；
  - Y是S的主码，是R的外码，则 T(R⋈S)= T(R)；
  - 所有S与R元组都有相同Y值，则 T(R⋈S)= T(R) *T(S)
- 对一般自然连接，做以下两个简化假设：
  - 值集包含
  - 值集保持

两关系自然连接大小的估计

- 令**V(R,Y)≤V(S,Y)**，假设**R**的每个元组**t**，与**S**中每个不同**Y**值等值匹配概率相同，即有**1/ V(S,Y)**机会与给定的一个**S**元组连接，获得元组数的期望值为**T(S) / V(S,Y)**。而**R**中共有**T(R)**个元组，故有：
  - T(R⋈S)= T(R) * T(S) / V(S,Y)
- 类似地，当**V(S,Y)≤V(R,Y)**时，有：
  - T(R⋈S)= T(R) * T(S) / V(R,Y)成立。
- 综合两者后，可得两关系自然连接的大小估计公式：
  - **T(R⋈S)= T(R) * T(S) / max{V(R,Y),V(S,Y)}**

(7.5_1)

例7.6

针对如下三个关系及其重要统计值，给出自然连接R⋈S⋈U的大小估值。

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⋈S)＝1000*2000/max{20,50}=40,000。

根据值集保持假设，V(R⋈S,c))＝100；

T((R⋈S) ⋈U)＝ 40000*5000/max{100,500}

= 400,000

两关系多连接属性的自然连接

- 对形如**R(X,Y1,Y2)⋈S(Y1,Y2,Z)** 连接，可从两关系单属性自然连接公式**7.5_1**推广得到如下的大小估计公式：

$$T(R \infty S) = \frac{T(R)T(S)}{\max\{V(R,Y1),V(S,Y1)\}\max\{V(R,Y2),V(S,Y2)\}}$$

- 例**7.7** 针对如下两个关系及其有关统计值，给出连接**R⋈cS**的大小估计值，其中，条件**c**为**R.b=S.d AND R.c=S.e**，属于一个等值连接条件。

R(a,b,c)	S(d,e,f)
T(R)=1000	T(S)=2000
V(R,b)=20	V(S,d)=50
V(R,c)=100	V(U,e)=50

先将等值连接转为自然连接处理，把R.b/S.d、R.c/S.e分别视为相同的属性，则该连接可归属为两关系的多属性连接。

- 估计代价：1000*2000/[max{20,50}* max{100,50}]=400

多关系自然连接结果大小 估计

例7.8　针对如下三个关系及其重要统计值，估计连接R(a,b,c)⋈S(b,c,d) ⋈U(b,e)的大小估计值。

R(a,b,c)	S(b,c,d)	U(b,e)
T(R)=1000	T(S)=2000	T(U)=5000
V(R,a)=100		
V(R,b)=20	V(S,b)=50	V(U,b)=200
V(R,c)=200	V(S,c)=100	
V(S,d)=400		
		V(U,e)=500

只有属性b出现3次，属性c出现2次

1000×2000×5000/[(50×200)×(200)]＝5000

7.5.3 消除重复操作的大小估计

- 若关系$R(a_1,a_2,\ldots,a_n)$是一个关系，其消除重复后的大小我们简单采用以下公式估算：
  - $\delta(R) = \min\{ 1/2, V(R, a_1), \ldots, V(R, a_n) \}$
  - 如果没有 $V(R, a_i)$ 类参数可用，我们就简单取 $\delta(R)$ 为1/2。
- 在典型优化器中，估计查询基本块大小只是采用了减少因子法，并没有用到连接操作的大小估计。

7.6 基于代价的枚举与优化

7.6.1 枚举候选计划

7.6.2 单关系查询优化

7.6.3 多关系查询优化

7.6 基于代价的枚举与优化

- 系统必须为每个查询定制一个赋值策略，生成一个执行计划。对于同一个查询，可能有几个执行计划都符合要求。
- 基于代价的优化器（Cost Based Optimizer，CBO)
  - 以代价大小作为衡量标准，选择代价最小的计划作为最终执行计划，并抛弃其它的执行计划。
  - 典型的优化器（如System R）中，基于代价的优化技术首先被独立应用到每个查询块。
- 优化器寻优过程至少包括两个基本步骤
  - 第一步，利用各种代数等价规则，为待赋值表达式--枚举可能的候选计划。（本节介绍）
  - 第二步，估计每个枚举计划的代价，作为选择计划的定量依据。

7.6.1 枚举候选计划

- 原则上，SQL优化器可以利用各种代数等价规则，系统地产生与给定代数表达式等价的表达式。
  - 所有可能的赋值计划，构成了所谓的赋值计划空间
- 为一个相对简单的查询探索非常大的候选计划空间可能会得不偿失。优化器通常不得不缩小它所考虑的候选计划空间。
  - 仅考虑所有可能赋值计划的一个较好子集，
  - 而不是不加选择地考虑所有代数等价计划。
- 可有效缩小需搜索计划空间的方法
  - 及时剪枝
  - 基于局部最优的剪枝技术。
  - 用启发式方法来减少需要考虑或探索的计划空间。

7.6.2 单关系查询优化

对应的代数表达式为：

$$\Pi_{S.rating,\ COUNT(*)}\ (\ \text{HAVING}_{\ COUNT\ DISTINCT\ (S.name)>2}\ (\ \text{GROUP BY}_{\ S.rating}\ (\Pi_{S.rating,\ S.name}(\ \sigma_{\ S.rating\ >5\ \wedge\ S.age=20}\ (Sailors)))))$$

- 例7.9 对每个职级大于5的水手分组，打印出职级和20岁水手的人数。要求每个组中至少有两个满足条件的不同名水手。SQL表达：

```
SELECT    S.rating, COUNT(*)  FROM  Sailors S
WHERE     S.rating>5 AND S.age=20
GROUP BY  S.rating
HAVING    COUNT DISTINCT (S.name)>2
```

(1)没有索引的计划

当无合适索引可用时，赋值计划比较确定：

- 先扫描关系，并应用σ和Π到每个被检索元组。
  - 扫描代价500 页；只输出<rating, sname>,减少因子取0.8; rating>5减少因子取0.5; age=20减少因子取0.1.
  - 输出大小：500 x 0.8 x 0.5 x 0.1 =20 页
- 最后阶段，附加处理GROUP BY、HAVING子句（如有的话）
  - 根据GROUP BY指定的排序字段进行分组排序
    - 用rating对临时关系进行排序，代价为 3 x 20 = 60 次I/O
  - 利用HAVING子句条件过滤分组（如有该子句）
  - 该计划总估代价：500 + 20 + 60 = 580。

（2）使用单索引存取路径

- 如果有几个索引能匹配**WHERE**子句的选择条件，它们都能提供一种可选的存取路径。
  - 优化器一般做法是：
    - **先选择具有最少估算代价的存取路径，**
    - **再应用投影，以及非主选择项（不与索引匹配的那部分选择条件）。**
    - **最后，再接着计算分组和聚合操作。**

(3) 使用多索引存取路径

- 前提
  - 有多个可匹配选择索引，且索引项是记录指针；
- 基本处理步骤
  - 用每个可匹配条件索引分别检索一组rids，并在主存中求这些*sids*组的交集，同时对rids交集进行基于page_id的排序；
  - 根据基于page_id排序的rids，检索满足所有匹配索引主选择项的元组；
  - 对检索到的元组，应用非主选择项和任何投影；
  - 最后处理分组、分组过滤和聚合操作。

(4) 使用排序索引存取路径

- 如果分组属性列表是一个B+树索引的前缀，则索引可被用来按分组所要求的顺序检索元组。
- 所有的选择条件可应用到每个被检索元组上，不用的字段可被移除，且每个分组的聚合操作可顺便被计算。
- 这个策略在有B+聚集索引情况下将工作得很好。

(5) 只用索引的存取路径

- 前提：查询中出现的所有属性，都已被包含在关系的一些稠密索引搜索键中。
- 这时可以不检索关系的实际元组。

例7.10

对于例7.7的查询，枚举出各种基于索引的赋值计划。假设索引项中都只包含记录指针，可用索引包括：

- *age*上的散列索引
- *rating*上的B+树索引
- 组合属性<*rating*,*sname*,*age*>上的B+树索引。试枚举出各种基于索引的赋值计划。

```
SELECT      S.rating, COUNT(*)
FROM        Sailors S
WHERE       S.rating>5 AND S.age=20
GROUP BY    S.rating
HAVING      COUNT DISTINCT (S.name)>2
```

7.6.3 多关系查询优化

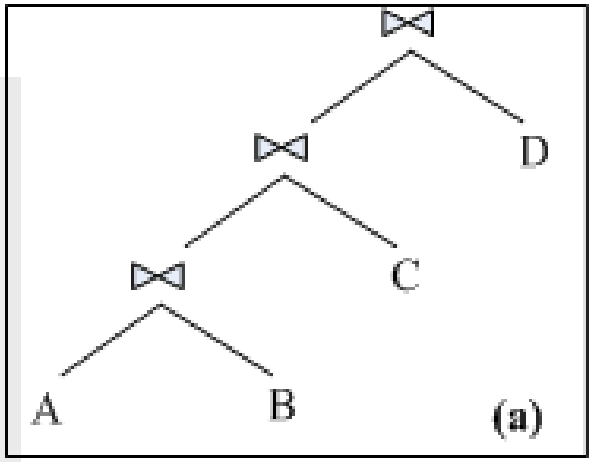
- 查询类：**FROM**子句中包含两个或更多关系，需要进行连接或叉积操作。
- 这类查询的赋值代价可能会很大，为它们寻求好的赋值计划非常重要。
- 被连接各关系出现的不同先后顺序，会使得中间关系大小迥异，从而可能导致代价差别显著的不同计划。

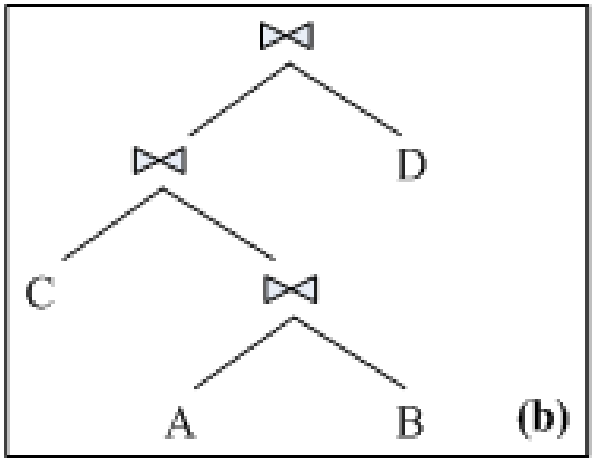
左深计划

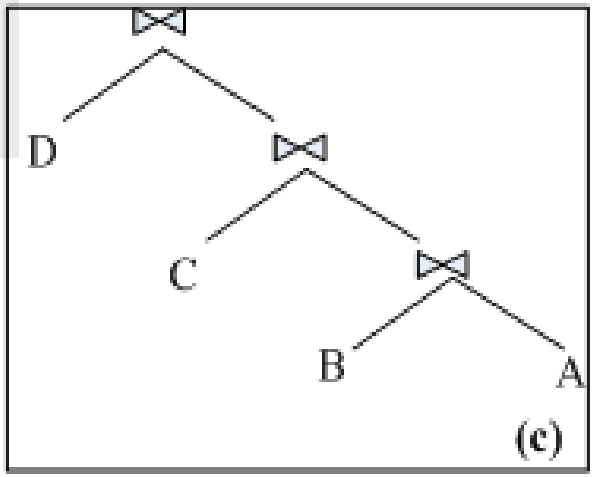
- 考虑一个形如A⋈B⋈C⋈D的四个关系自然连接查询 的几个等价关系代数操作树。

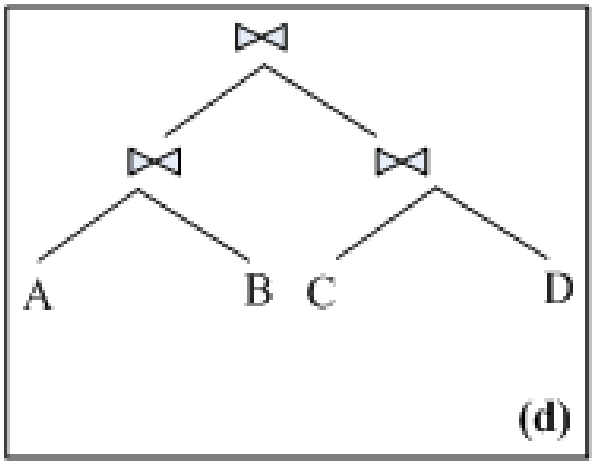
图中仍遵循如下基本约定：

- 左子节点是连接外层关系
- 右子节点是内层关系的。









算法7.1 多关系连接的多阶段枚举算法

- 阶段1：分别以每个单关系作为左深树最左关系，枚举最便宜计划和每种元组排序的最便宜计划。
- 阶段2：用阶段1保留下来的每个计划中的关系作外层关系，用另外的其它单关系作内层关系，枚举各种组合的两关系连接计划。
- 阶段3：枚举所有三个关系组合的计划。类似阶段2的处理，只不过这时将用阶段2保留下来的各种两关系计划作为外层关系。
  - 其它额外阶段：当有3个以上更多关系时，可按此类似方式进行处理，直到最终产生包含查询中所有关系的计划。
- 最后阶段：若多关系查询有聚合汇总操作，则最后还需为这类附加操作确定最便宜的处理方案。

算法7.1应用--例7.11

- 例7.11 对查询：

```
SELECT S.sname
FROM  Reserves R, Sailors S
WHERE R.sid = S.sid AND R.bid=100 AND S.rating > 5
```

假设有下面的索引（都是非聚集的且索引项中只含键/记录指针）可用：

- Sailors:*rating* 上的B+树
- Sailors:*sid* 上的散列索引
- Reserves:*bid* 上的B+树索引。

试枚举左深树风格计划。

例7.12

针如下包含三个关系连接的查询：

```
SELECT  S.sid, COUNT(*) as numres
FROM    Sailors S, Reserves R, Boats B
WHERE   R.sid = S.sid AND B.bid=R.bid
          AND B.color='red'
GROUP BY  S.sid
```

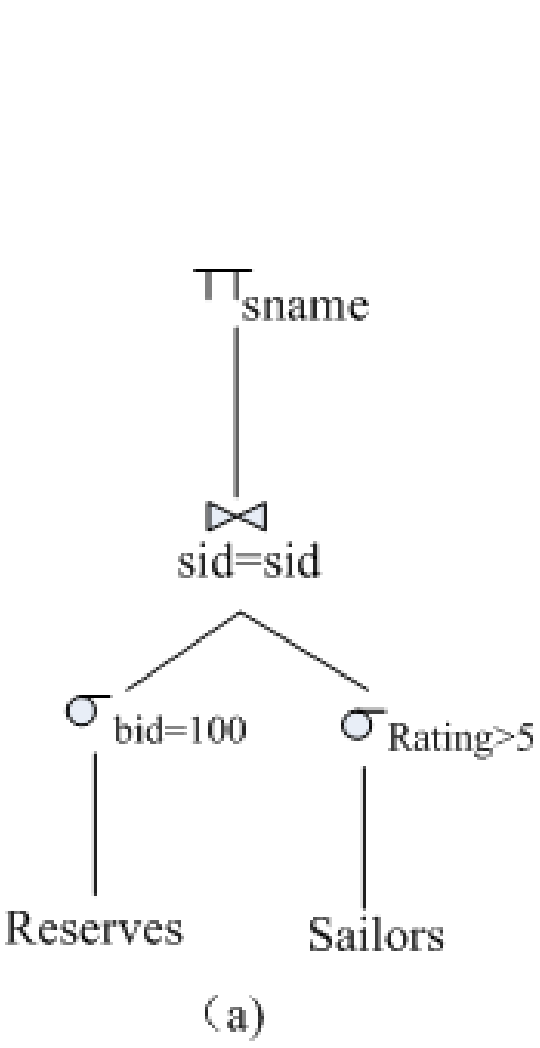
假设有以下的索引可用：

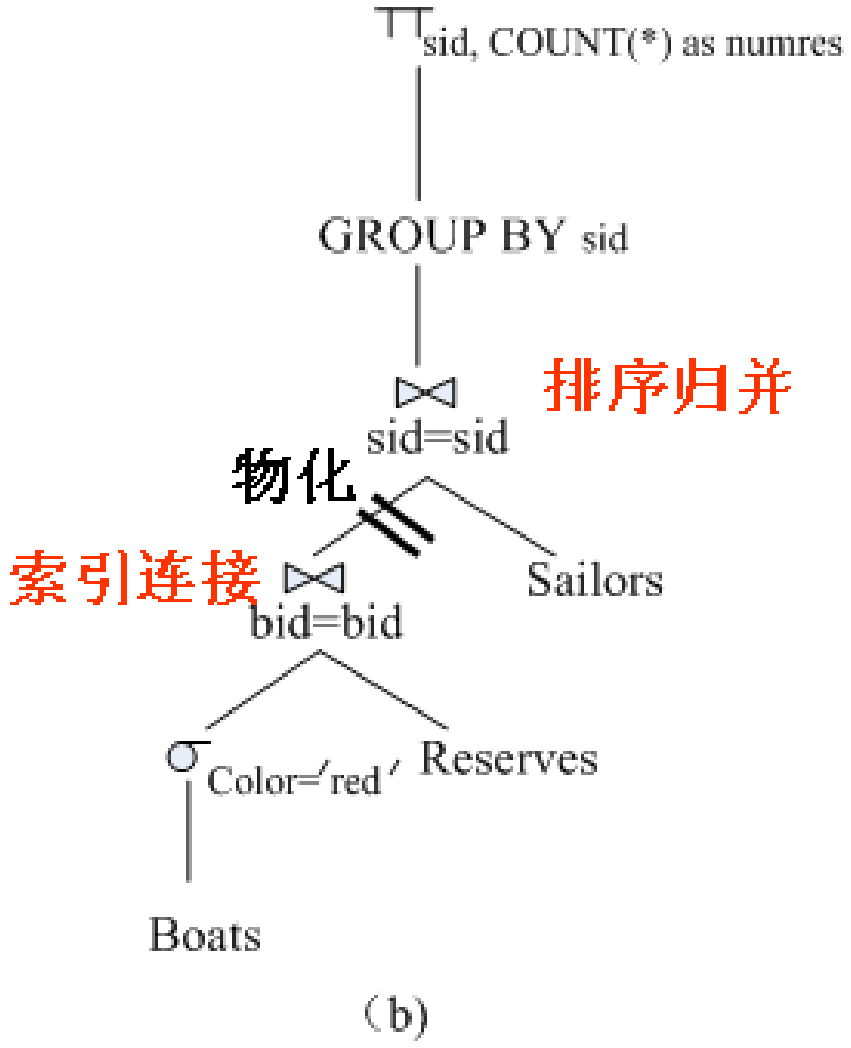
- Reserves:*sid*上的B+树、Reserves:*bid*上的聚集B+树；
- Sailors:*sid*上的一个B+树索引和一个散列索引；
- Boats:*color*上的一个B+树索引和一个散列索引。

试给出一个能有效计算该查询的计划。

例7.10

- 最终可得到如下图(b)所示的查询赋值计划树





7.7 *处理嵌入子查询

- 考虑下面查找具有最高职级水手的嵌入查询表达：

  例7.11 SELECT *S.sname* FROM Sailors *S*
  WHERE *S.rating* =
  (SELECT *MAX(S2.rating)* FROM Sailors S2)

- 在这个语句中，嵌入子查询只需要被赋值一次，产生一组简单的值。这组简单值随后以作为原查询语句的一部分被顶层查询。
  - 若水手最高工龄为8，则相当于原WHERE子句被修改为WHERE *S.rating*=8。
- 子查询也可能返回一个元组集，或者更准确说，是一个**SQL**意义上的关系表（可能还有重复元组）。

7.7 *处理嵌入子查询

- 例7.12 查找曾经被指派到102号船值勤的所有水手，输出这些水手名字。

```
SELECT S.sname FROM Sailors S
WHERE S.sid IN
    (SELECT R.sid FROM Reserves R WHERE R.bid=102)
```

- 这个语句的嵌入子查询也可只用一次执行完成赋值，产生一组***sids***集。
  - 对Sailors的每个元组，系统都必需检查其*sid*值是否在这组*sids*集中。
  - 这个检查相当于限定了Sailors与这组*sids*值集的连接。原则上，我们可选择各种连接方法。
  - 通常是选择以这组非排序的*sids*集作为连接的内层关系，来进行嵌套循环连接

7.7 *处理嵌入子查询

- 典型优化器处理嵌入子查询的最常用方法
  - 使用变体技术，将它转化为连接来处理。
  - 当主查询与嵌入子查询没有关联，嵌入子查询只需要执行一次时，这种变体技术一般都能适用。
- 但如果主查询与嵌入子查询相互关联，即嵌入子查询为关联子查询时，简单的变体技术就失效了

例如，查询

```
SELECT  S.sname  FROM    Sailors S
WHERE  EXISTS (SELECT * FROM Reserves R
        WHERE R.bid=102 AND S.sid=R.sid);
```

- 目前，关于嵌入子查询处理，是关系查询优化器的一个薄弱环节，优化方法不多或很有限。
- 很多优化器，如**System R**及其后续的**Starburst**，试图重写**SQL**查询，只要有可能就把子查询转换成连接；把关联嵌入子查询转为没有关联的嵌入子查询。
  - 但大多数现代优化器尚不能有效进行这类转换，这就要求用户在构造查询表达式时，尽可能给出非嵌入的查询表达，以便典型优化器发现更好的赋值策略。
- 对不能转换的非关联嵌入子查询，则将子查询就作为独立的表达式，并独立地进行优化。
- 最坏情况是按前述原始方法,处理关联嵌入子查询。

7.8 *Oracle优化器简介

7.8.1 基于规则的优化器

7.8.2 Oracle基于代价的优化器

7.8.3 如何设定Oracle优化器的工作模式