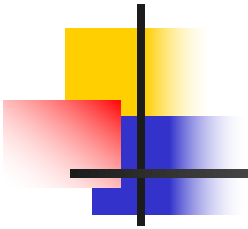
第6章 查询处理和优化

6.1 引言

查询处理—— 从查询语句出发，到获得查询结果的处理过程。

查询优化—— *DBMS*对描述性语言表达的查询语句进行分析，为其确定合理、有效的执行策略和步骤的过程。

查询优化是**查询处理**中的重要一环，对关系*DB*尤其如此。



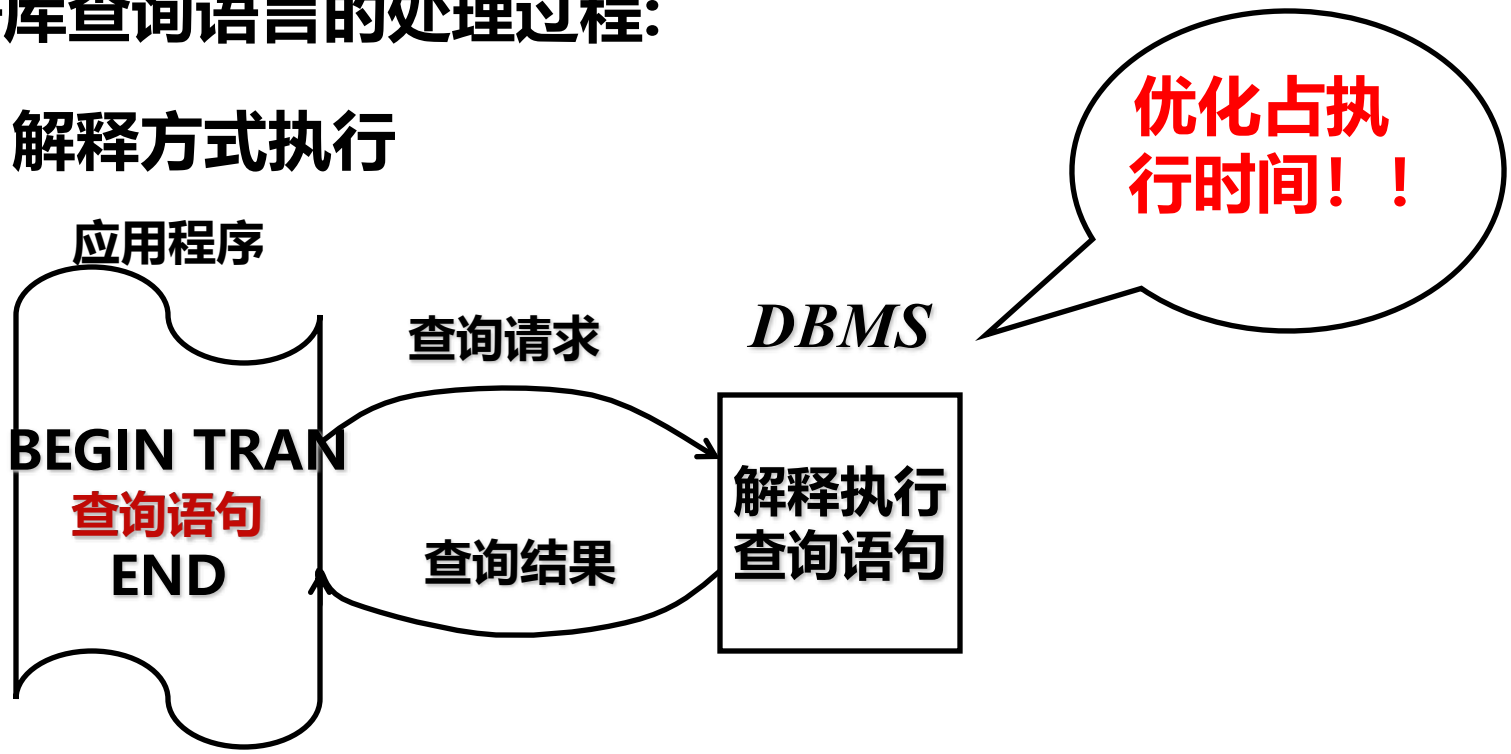
例如：12+64+88=?

（12+88）+64=164

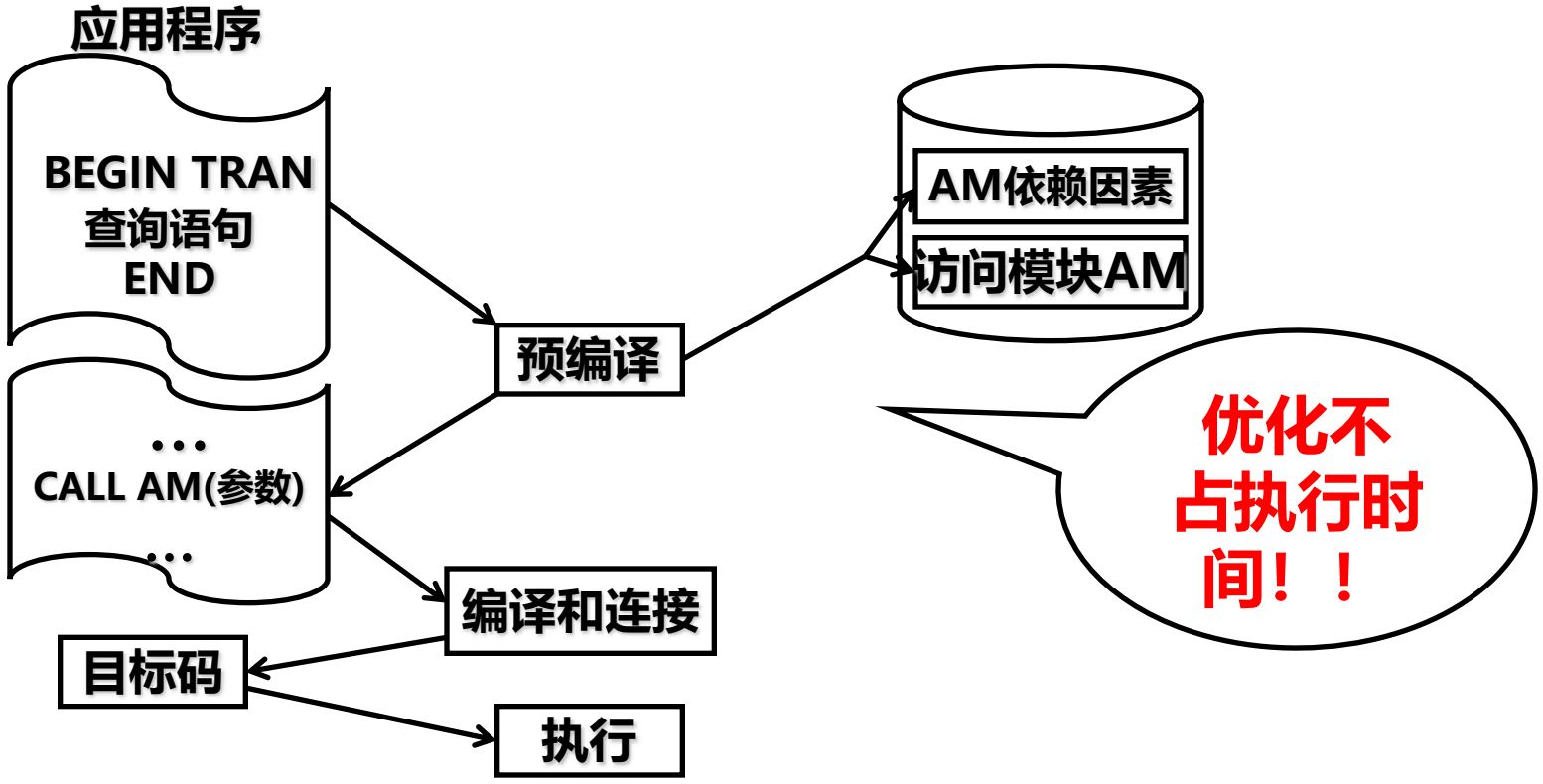
查询优化是相对而言的，可能的执行策略很多，穷尽代价很大，不能片面追求绝对的最优。

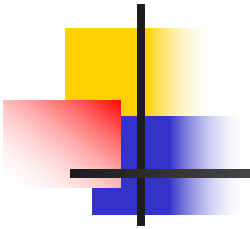
数据库查询语言的处理过程:

（1）解释方式执行



(2)编译方式

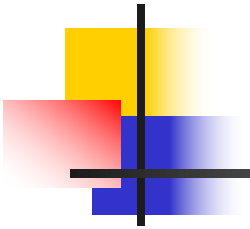

应用程序
BEGIN TRAN
查询语句
END
...
CALL AM(参数)
...
预编译
AM依赖因素
访问模块AM
优化不
占执行时
间！！
编译和连接
目标码
执行



解释方式和编译方式各适用于什么情况?

对于常见的例行事务，编译方式可提高性能。

对于简短的即时查询，解释方式灵活实用。



- **代数优化** 对查询语句进行变换不涉及存取路径
- **物理优化** 根据存取路径选择合理的存取策略进行优化
- **规则优化** 仅根据启发式规则选择执行的策略进行优化
- **代价估算优化**

6.2 代数优化

代数优化对查询进行等效变换，以减少执行开销。

代数优化的原则是**尽量减小查询过程中间结果的大小**。

选择、投影操作通常能够有效地减小关系的大小。连接、迪卡尔乘积和并操作容易生成较大的查询中间结果。

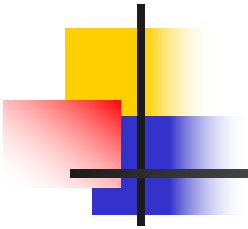
因此，**先做选择、投影**；**先做小关系间的连接，再做大关系的连接**；甚至需要先找出查询中的公共表达式，以避免重复运算。

常用变换规则

1. $\sigma_{c1\ AND\ c2 \ldots AND\ cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots))$
2. $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$
3. $\prod_{list_1}(\prod_{list_2}(\ldots(\prod_{list_n}(R))\ldots)) = \prod_{list1}(R),$

   $list_1 \subseteq list_2 \subseteq \ldots \subseteq list_n$
4. $\prod_{A1,A2,\ldots,An}(\sigma_c(R)) \equiv \sigma_c(\prod_{A1,A2,\ldots,An}(R))$

   $Attr(C) \subseteq \{A1, A2, \ldots, An\}$



5. **R JN S = S JN R**

6. $\sigma_c(R \text{ JN } S) \equiv (\sigma_c(R)) \text{ JN } S,$

   $\text{Attr(c)} \subseteq \text{Attr(R)}$

7. $\sigma_{c1\ AND\ c2}(R\ JN\ S) \equiv (\sigma_{c1}(R))JN(\sigma_{c2}(S)),$

   $Attr(c1) \subseteq Attr(R), Attr(c2) \subseteq Attr(S)$

8. 属性集 $\mathrm{L}=\{\mathrm{A}_1,\ldots,\mathrm{A}_n,B_1,\ldots,B_m\}$,

其中, $\{\mathrm{A}_1,\ldots,\mathrm{A}_n\}\subseteq Attr(R),\{B_1,\ldots,B_m\}\subseteq Attr(S)$

则 $\prod_L(R\ JN_c\ S)\equiv(\prod_{A1,\ldots,An}(R))JN_c(\prod_{B1,\ldots,Bm}(S))$

式中 $Attr(c)\subseteq L$

9. $\theta\in\{\cup,\cap,-\}$则 $\sigma_c(R\,\theta\,S)\equiv(\sigma_c(R))\,\theta\,(\sigma_c(S))$

10. $\theta\in\{\cup,\cap,-\}$则 $\prod_L(R\,\theta\,S)\equiv(\prod_L(R))\,\theta\,(\prod_L(S))$

11. $\theta \in \{JN, \times, \cup, \cap\}$则，

$$(R\ \theta\ S)\ \theta\ T \equiv R\ \theta\ (S\ \theta\ T)$$

注意：规则11中，对于连接运算，可能出现S与T之间无连接条件的情况，此时的连接运算成为迪卡尔乘积。

例如：(R JN_{c1} S)JN_{c2} T,

式中，Attr.(c1) $\subseteq$ Attr.(R) $\cap$ Attr.(S)

Attr.(c2) $\subseteq$ Attr.(R) $\cap$ Attr.(T)

而S和T之间没有连接条件。可改写为：

R $JN_{c1\ AND\ c2}$ (S$\times$T)

范例p118例6-1

设有S(供应商)，P(零件)，SP(供应关系)三个关系，关系模式如下：

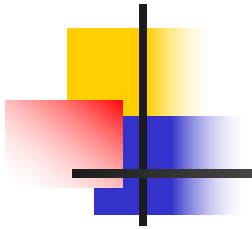
S(SNUM,SNAME,CITY)

P(PNUM,PNAME,WEIGHT,SIZE)

SP(SNUM,PNUM,DEPT,QUAN)

有如下查询

```
Q：SELECT SNAME
     FROM S,P,SP
     WHERE S.SNUM = SP.SNUM
       AND   SP.PNUM = P.PNUM
       AND   S.CITY = ‘NANJING’
       AND   P.PNAME = ‘BOLT’
       AND   SP.QUAN > 10000
```



- **SQL语句转化为原始查询树**

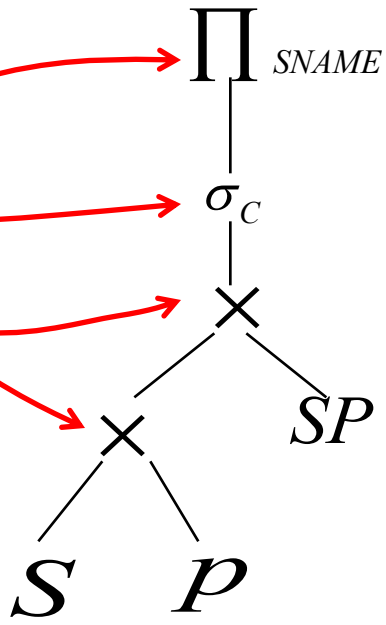
Select $\longrightarrow \prod$

From $\longrightarrow \times$

Where $\longrightarrow \sigma$

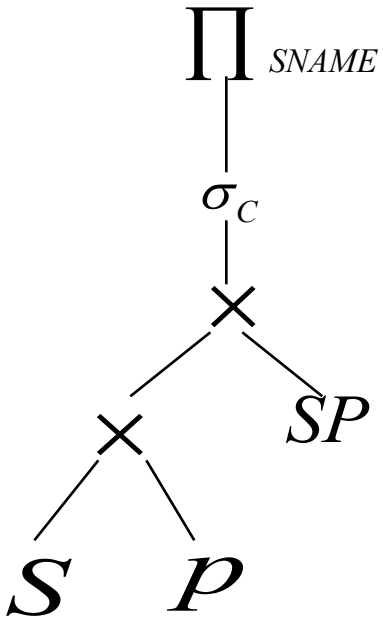
Q可用右图所示的原始查询树表示：

```
Q :  SELECT SNAME
     FROM S,P,SP
     WHERE S.SNUM = SP.SNUM
       AND   SP.PNUM = P.PNUM
       AND   S.CITY = 'NANJING'
       AND   P.PNAME = 'BOLT'
       AND   SP.QUAN > 10000
```



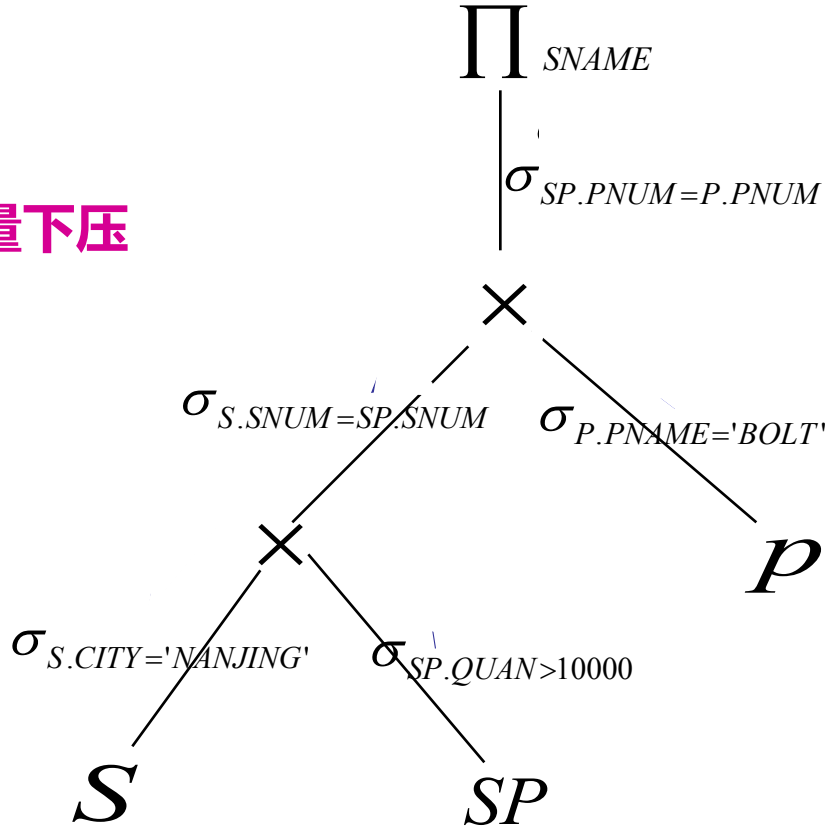
原始查询树

选择操作下压



原始查询树

选择操作尽量下压



先连接小关系

S,P,SP经选择后得S’、P’、SP’，估算大小：

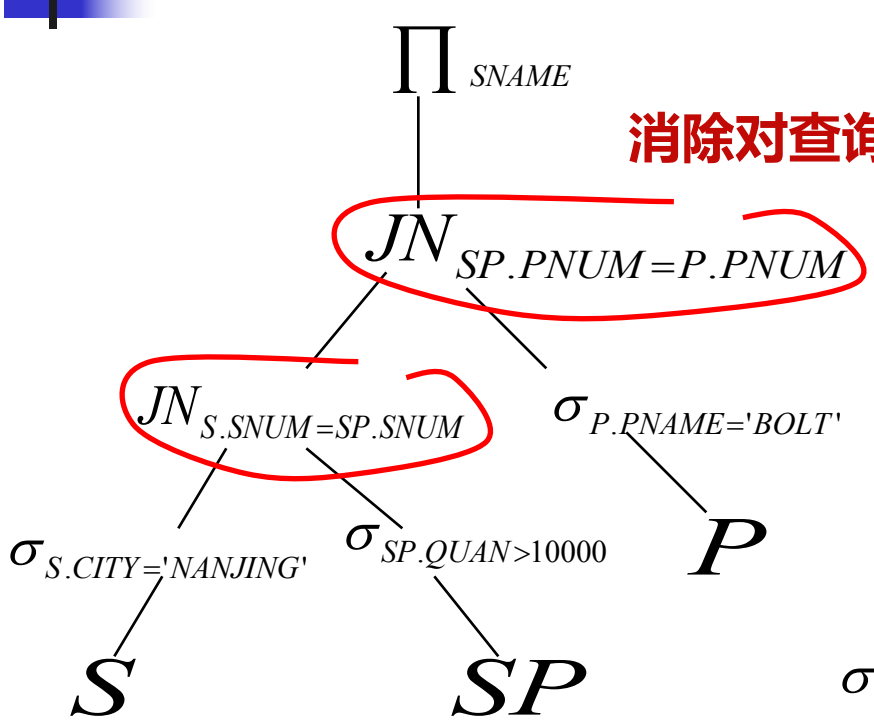
$|S'|=|S|/N_{CITY}$

$|P'|=|P|/N_{PNAME}$

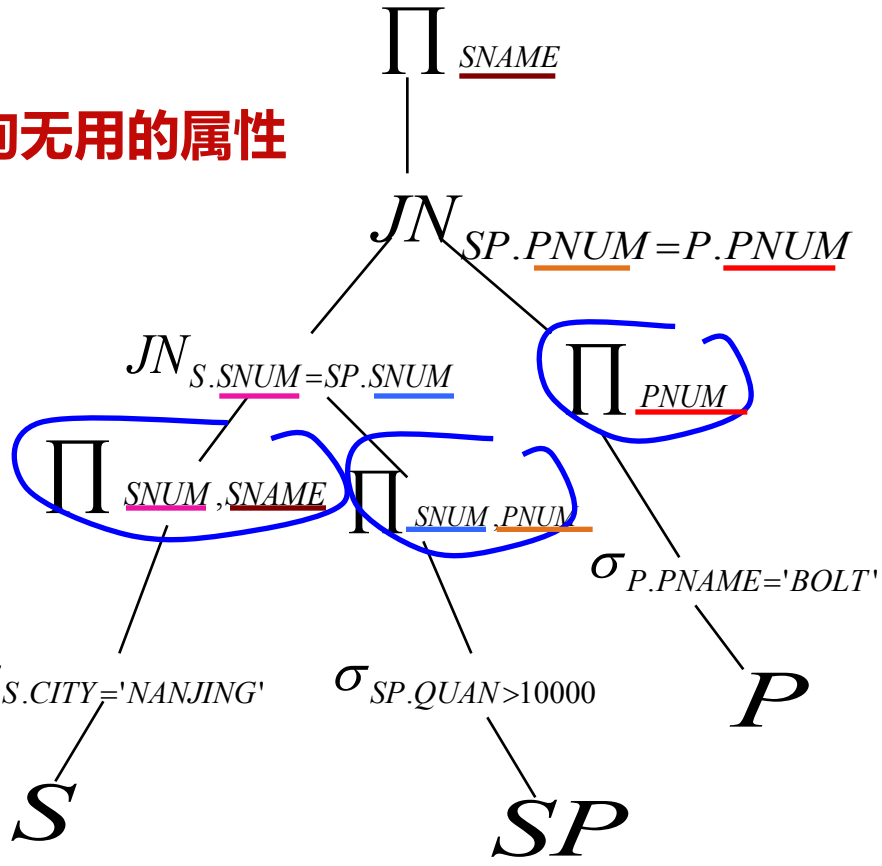
$|SP'|=|SP|\times(V_{max}\text{-}10000)/(V_{max}\text{-}V_{min})$

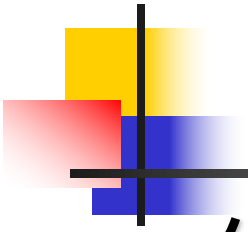
设$|S'|<|P'|$, $|SP'|<|P'|$

消除对查询无用的属性



消除对查询无用的属性





代数优化的基本步骤：

1.以SELECT子句对应投影操作，以FROM字句对应迪卡尔乘积，

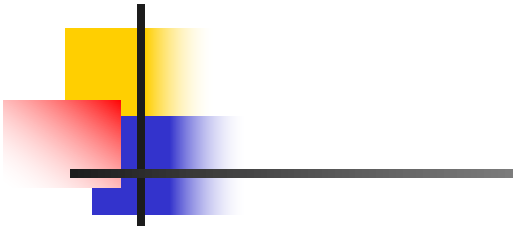
以WHERE子句对应选择操作，生成原始查询树。

2.应用变换规则2)、6)、7)、9)、10)，尽可能将选择条件移向树叶方向。

3.应用连接、迪卡尔乘积的结合律，按照小关系优先做的原则，重新安排连接（笛卡尔乘积）的顺序。

4.如果笛卡尔乘积后还须按连接条件进行选择操作可将两者组合成连接操作。

5.对每个叶节点加必要的投影操作，以消除对查询无用的属性。



以上所讨论的都是非嵌套查询。嵌套查询比较复杂，分如下两种情况：

1.嵌入的查询块与上层查询无关

从最低层查询开始，用上述步骤和规则，逐层计算各查询快所等效的关系，直至求出查询结果。

2.嵌入的查询块与上层查询有关

一般用代入法。

例如：

```
SELECT  A1
FROM    R1
WHERE  R1.A2 比较符 CONST1
       AND R1.A3 IN
           (SELECT A4
            FROM   R2
            WHERE R2.A5 比较符 R1.A1)
```

将第一层查询所涉及R1表中的每条记录，代入虚线框所标出查询体，**此时R1.A1为一个常量值**，判断该记录是否满足查询条件。

存在什么问题?
能否再进行优化?

SELECT

看作临时表*R1'*

FROM R1'

ND

ANI WHERE 3 IN

R1'.A3

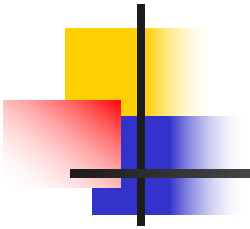
(SELECT A4

FROM R2

WHERE R2.A5 比较符 R1.A1)

将*R1'*的每个元组逐个代入，检查限制条件是否满足，以减少需检查的上层查询所涉及表的元组数目！

注意：采用代入法时，尽可能作“部分选择”！



有些DBMS将嵌套查询转换为等效的非嵌套查询，但是这种方法不一定在所有情况下都可行。

6.3 依赖于存取路径的规则优化

代数优化不涉及存取路径，对各种操作的执行策略无从选择。只能在操作的次序和组合上做一些变换和调整。

单靠代数优化比较粗糙，优化效果有限，合理选择存取路径，往往能收到显著的效果。

本节结合存取路径的分析，讨论各种基本操作执行的策略及其选择原则。

6.3.1 选择操作的实现和优化

选择操作的执行策略与选择条件、可用的存取路径以及选取的元组数在整个关系中所占的比例有关。

选择条件：

- 等值 =
- 范围 >,<
- 集合 IN,EXISTS,NOT EXISTS
- 复合 AND,OR

- 实现方法：顺序扫描、尽量利用散列索引等方法。

选择操作选择存取路径的启发式规则：

（1） 对于小关系，顺序扫描。

（2） 若无索引、散列等存取路径可用，或估计选取的元组数占关系的比例较大（大于20%）且有关属性上无簇集索引，顺序扫描。

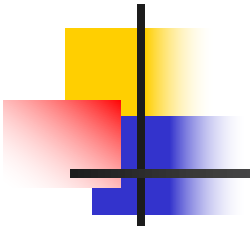
（3） 对于主键的等值选择，优先选用主键的索引或散列。

（4） 对于非主键的等值选择，若选取的元组数占关系的比例较小（小于20%），可以用无序索引；否则只能用簇集索引或顺序扫描。（为什么？）

（5）.对于范围条件，先通过索引找到范围的边界，再通过索引的顺序集沿相应方向搜索，如中选的元组数在关系中所占比例较大，宜采用簇集索引或顺序扫描。

（6）对于用AND连接的合取选择条件：

- 优先选用多属性索引
- 若有多个可用的次索引，可用预查找处理，最后做其余条件检查
- 个别条件可用（3）（4）（5）之一，求得相应组，再将这些元组用其它条件筛选
- 顺序扫描



（7）用OR连接的析取选择条件，尚无好的方法。只能按其中各个条件分别选出一个元组集，再求这些元组集的并。

在OR连接的诸条件中，只要有一个条件无合适的存取路径，就只能用顺序扫描！

6.3.2 连接操作的实现和优化

连接开销较大，为查询优化的重点，这里主要讨论二元连接（Two Way Join）。

实现方法

1.嵌套循环法（nested loop）

2.利用索引或散列寻找匹配元组法

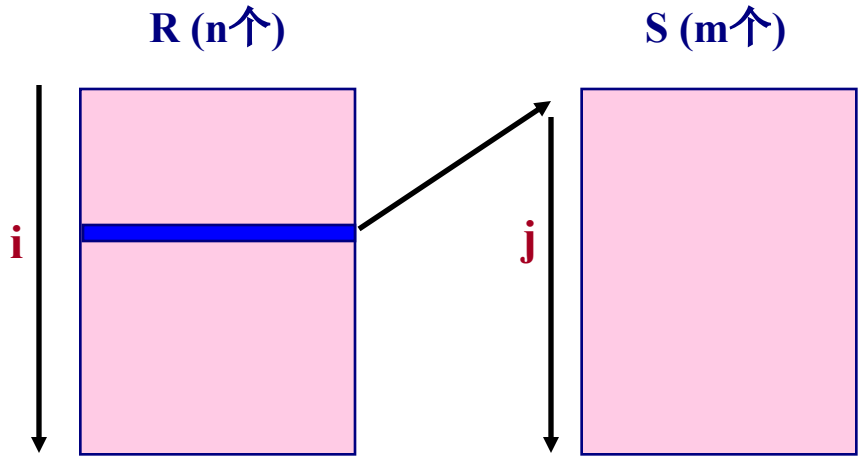
3.排序归并

4.散列连接法

1).嵌套循环

关系R与S进行连接操作，最原始的办法是取R的一个元组，与S的所有元组比较，凡是满足连接条件的元组就进行连接并且作为结果输出。然后再取R的下一个元组，和S的所有元组比较，直到R的所有元组比较完为止。

$$R \bowtie_{R.A=S.B} S$$



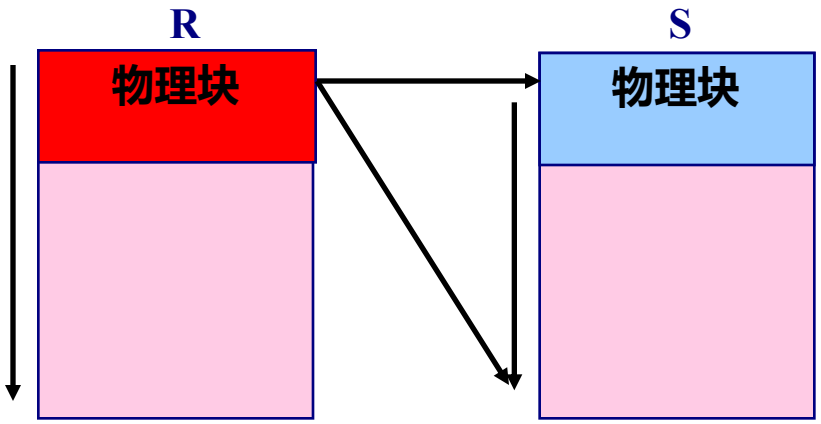
嵌套循环算法

```
/*设R有n个元组，S有m个元组*/
i:=1,j:=1;
while(i≤n)
do{while(j≤m)
    do{if R(i)[A] = S(j)[B]
          then 输出<R(i),S(j)>至T;
          j := j + 1
        }
 j:=1,i:=i+1
}
```

T为R和S连接的结果

R为外关系（outer relation），
S为内关系（inner relation）。

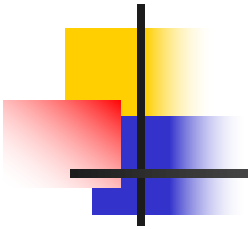
事实上，**关系是以物理块为单位取到内存**，设R和S各有一缓冲块，P_R为R的块因子（每块中所含的元组数）。则R每次I/O取P_R个元组，可改进上述算法，使S扫描一次可以与R的P_R个元组比较，那么S的扫描次数为$b_R=[n/P_R]$。



假设，b_R和b_S分别为关系R和关系S占用物理块的数目($b_R=[n/P_R]$)，n_B为可供连接使用的缓冲块数。若将其中的n_B-1块作为外关系缓冲块，1块作为内关系缓冲块。

则以R为外关系、S为内关系，用嵌套循环法进行连接所需访问的物理块数为$b_R+[b_R/(n_B-1)]*b_S$，对应最小I/O值。

问题：增加外关系R的缓冲块（每次多取几块R的数据）或增加内关系S的缓冲块都能减少I/O次数。为什么将n_B-1块作为外关系缓冲块，1块作为内关系缓冲块，是最优分配策略?



问题：嵌套循环法进行连接操作，以R为外关系、S为内关系;还是以S为外关系、R为内关系所需I/O次数更少？作为外层循环的关系，有什么要求?

应将占用物理块少的关系，作为外关系!

2).利用索引或散列寻找匹配元组法

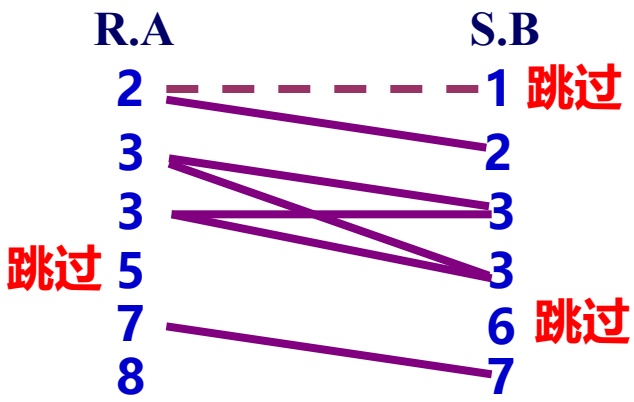
在嵌套循环法中，内关系上要做多次顺序扫描，若内关系上有合适的存取路径（连接属性上的索引散列等），可以避免内关系上的顺序扫描，以减少I/O次数。

问题：若在内关系的连接属性上建有索引？是否一定能够提高内关系和外关系的匹配效率?

当每次循环所选的匹配元组数在内关系中占有较大比例（例如超过15%）时，用无序索引甚至还不如用顺序扫描的方法。内关系的连接属性上有簇集索引时，索引对减少连接所需I/O次数的作用最明显。

3).排序归并

如果R和S按连接属性排序，可按序比较R.A和S.B以找出匹配元组。

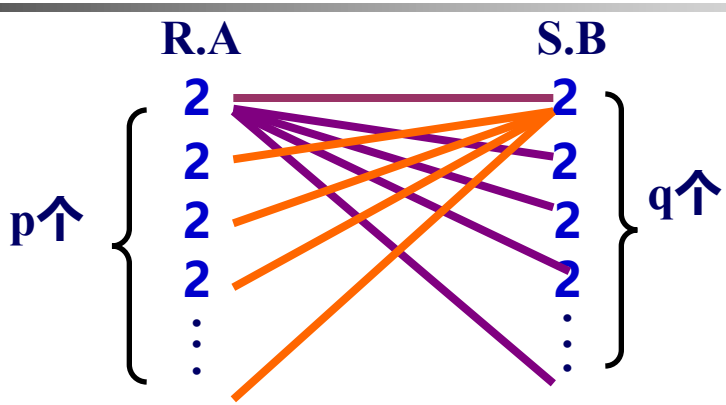


算法：

```
R按属性A排序  /*设R有n个元组*/
S按属性B排序 /*设S有m个元组*/
i←1,j←1;
While(i≤n)and(j ≤m)
    do{if R(i)[A]>S(j)[B]
          then j←j+1
          else if R(i)[A]<S(j)[B]
          then i←i+1
          else{
          /* R(i)[A]=S(j)[B]，输出连接元组*/
```

```
输出< R(i),S(j) >至T;
/*输出R(i)和S中除S(j)外的其他元组所组成的连接元组 */
l←j+1;
While(l≤m) and (R(i)[A]=S(l)[B])
    do{输出< R(i),S(l) >至T;
        l←l+1;}
/*输出S(j)和R中除R(i)外的其他元组所组成的连接元组 */
k←i+1;
While(k≤n) and (R(k)[A]=S(j)[B])
    do{输出< R(k),S(j) >至T;
        k←k+1;}
i←i+1,j←j+1;}
}
```

问题：等值匹配对使用排序归并法进行连接操作的效率有什么影响?



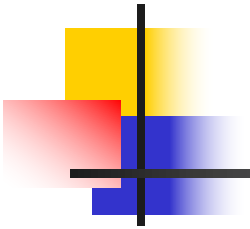
注意等值的扫描次数（假设p≤q）：

[1+(q-1)+(p-1)]+[1+(q-2)+(p-2)]+…

+[1+(q-p)+(p-p)]

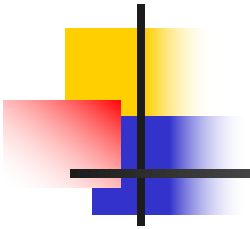
=[(p+q-1)+(p+q-2q+1)]/2*p

=p*q ⇔ O(pq)



4).散列连接法

连接属性R.A和S.B应具有相同的值域，用相同的散列函数，把R和S散列到同一散列文件中。符合连接条件的元组必然在同一通中（注意：同一桶中的元组未必都满足连接条件）。只需把桶中的匹配元组取出即可获得连接结果。



关键在于建立一个供连接用的散列文件。

建立散列文件需要对R、S各扫描一次，且关系R和S一般不会对连接属性进行簇集。故而，每向散列文件加入一个元组，都需要一次I/O操作。

如何减少I/O次数?

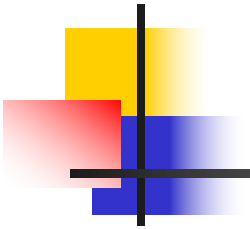
可以在桶（散列文件）中不填入R、S的实际元组，而是代之以元组的tid，从而大大的缩小散列文件，使其有可能在内存中建立，而仅需对R、S各扫描一次。

扫描R和S时，取出$\Pi_A(R)$、$\Pi_B(S)$，附在相应的tid后，连接时以桶为单位，按$\Pi_A(R)=\Pi_B(S)$找出匹配元组的tid对。

问题：似乎多此一举！匹配元组的tid一定在同一桶中！为什么还要按$\Pi_A(R)=\Pi_B(S)$找出匹配元组？这么做有必要么？

注意：$A=B \Rightarrow h(A)=h(B)$，但不一定有：

$h(A)=h(B) \Rightarrow A=B$



在取实际元组时，为减少物理块访问，可将各桶中，匹配元组的tid按块分类，一次集中取出同一块中所需的所有元组，当然这需要较大的内存开销。

连接方法的启发式规则

1）两个关系都已按连接属性排序，则优先用排序归并法；两个关系中已有一个关系按连接属性排序，另一个关系较小，也可先对未排序关系按连接属性排序，再用排序归并法。

2）两个关系中有一个关系在连接属性上有索引（特别是簇集索引）或散列，可以另一关系为外关系，顺序扫描，并利用内关系上的索引或散列寻找其匹配元组，以代替多遍扫描。

3）不具备上述条件且关系较小，可用嵌套循环法。

4）不具备1，2，3规则，可用散列连接法。

6.3.3 投影操作的实现

一般与选择、连接同时进行，无需附加的I/O开销。

若投影属性集中不包含主键，则投影结果中可能出现重复元组。

消除重复元组可以用排序或散列等方法。

散列法：将投影结果按某一属性或多个属性散列成一个文件，当一个元组被散列到一个桶中时，可检查是否与桶中已有元组重复。

用排序法消除重复元组

对关系R的每个元组t，生成t[<投影属性集>]，并存于T’中；/* T’ 是未消除重复元组的投影结果*/

If <投影属性集>含有R的主键

then T←T’

else{T’按所有属性排序；

i←1,j←2;

while(i≤n)

do{输出元组T’(i)到T;

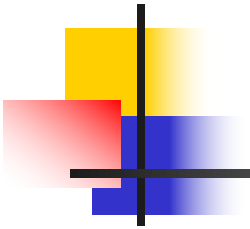
```
while T'(i)= T'(j) do j←j+1;
/*消除重复元组，设有伪元组T'[n+1]≠T'[n]*/
i←j,j←i+1;
  }
}
```

6.3.4 集合操作

常用集合操作：笛卡尔乘积、并、交、差等。

笛卡尔乘积将两个关系的元组无条件地互相拼接，**一般用嵌套循环法实现，做起来很费时，结果要比参与运算的关系大的多。应尽量少用！**

设关系R、S并兼容，对R、S进行并（交、差）操作，可以先将R和S按同一属性（通常选用主键）排序，然后扫描两个关系，选出所需的元组。



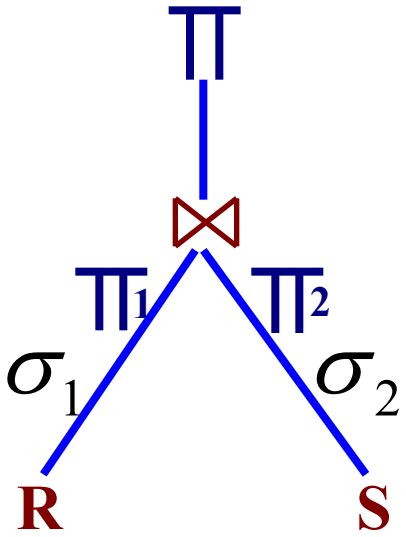
散列是上述并交差操作的另一种求解方法：

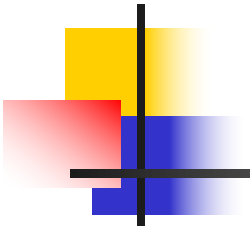
将关系R散列到一个散列文件中，再将S散列到同一文件中。同时检查桶中有无重复元组。对于并，不再插入重复元组；对于交，选取重复元组；对于差，从桶中取消与S重复的元组。

6.3.5 组合操作

有时，多个操作组合起来同时进行，如投影和选择操作组合起来执行（消除重复元组另外单独进行），可提高效益。

还可以在更大范围内，将多个操作组合起来执行。





假设连接用嵌套循环法，R为外关系，S为内关系，R的选择、投影可在扫描R时执行，S的选择、投影可在首次扫描S时执行，并将选择、投影的结果存入临时文件，之后各轮只需扫描临时文件即可。

最后一个投影操作，可在生成连接结果的同时进行。

6.5 结束语

在执行前进行优化称为**静态优化**，只能利用统计数据，有时不一定准。

在查询执行时进行优化称为**动态优化**，用实际执行结果估算代价，比较符合实际，但每次执行都要优化，不适于编译实现，也增加了执行时间。只能利用统计数据，有时不一定准。另外，**优化时，要等待中间结果，增加了等待时间和数据的相关性，不利于并行性**。