# 第四部分：查询处理

韩丽萍
计算机学院
联系邮件：liping@njupt.edu.cn

# 查询处理

Q （select...from...where....）

→ Query Plan （查询计划）

针对：非过程性的SQL语句声明转化成具体的执行过程，从而取得查询结果

# 一个稍复杂的查询例子

R(A, B, C)

S(C, D, E)

Select B,D

From R,S

Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C

R

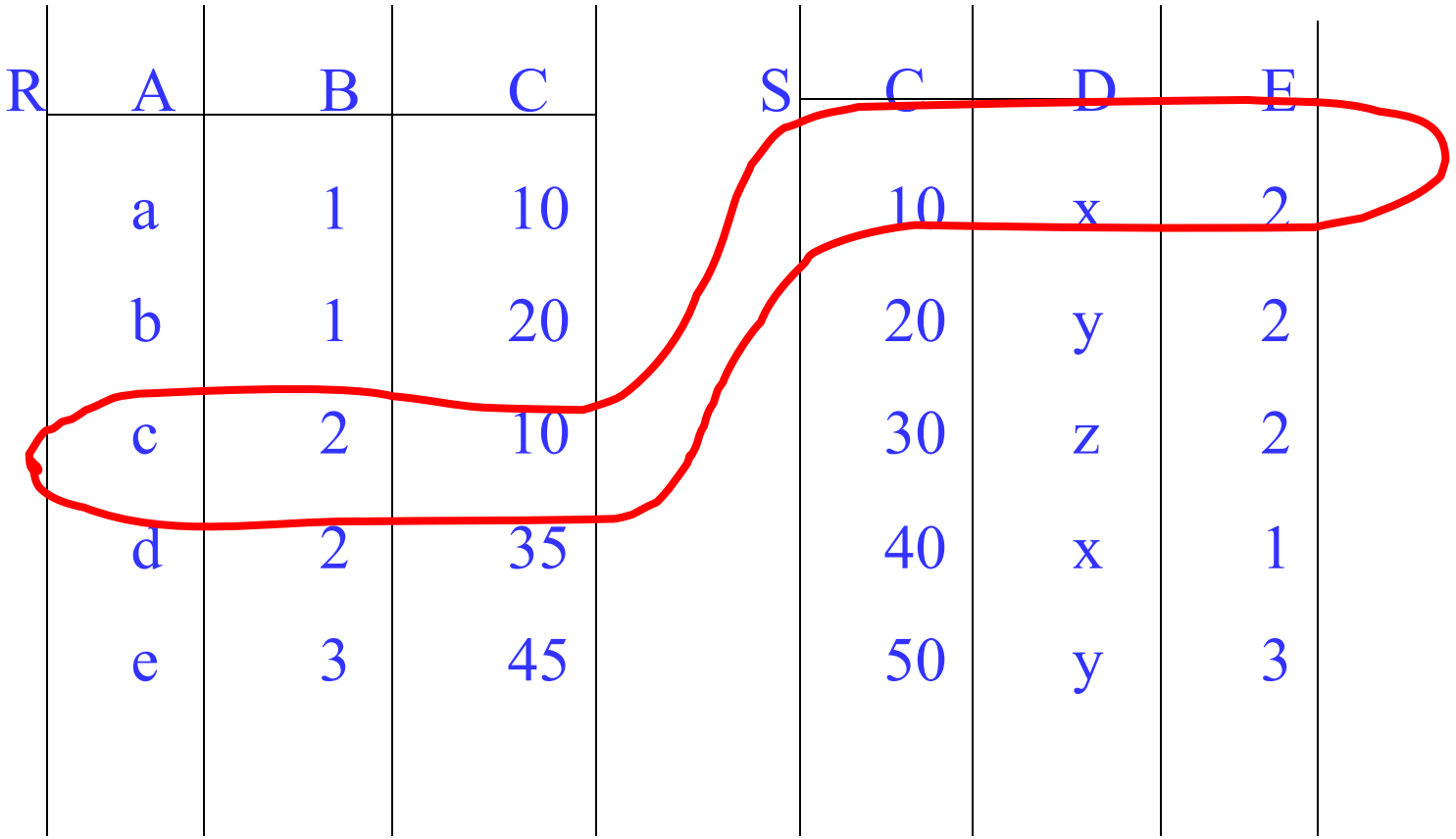
| A | B | C |
|---|---|---|
| a | 1 | 10 |
| b | 1 | 20 |
| c | 2 | 10 |
| d | 2 | 35 |
| e | 3 | 45 |

S

| C | D | E |
|---|---|---|
| 10 | x | 2 |
| 20 | y | 2 |
| 30 | z | 2 |
| 40 | x | 1 |
| 50 | y | 3 |

Select B,D From R,S

Where R.A = “c” $\wedge$ S.E = 2 $\wedge$ R.C=S.C



| R | A | B | C |
|---|---|---|---|
| | a | 1 | 10 |
| | b | 1 | 20 |
| | c | 2 | 10 |
| | d | 2 | 35 |
| | e | 3 | 45 |

| S | C | D | E |
|---|---|---|---|
| | 10 | x | 2 |
| | 20 | y | 2 |
| | 30 | z | 2 |
| | 40 | x | 1 |
| | 50 | y | 3 |

Answer

| B | D |
|---|---|
| 2 | x |

# 如何进行查询处理?


- 笛卡尔积
- 选择元组
- 进行投影

RXS

| R.A | R.B | R.C | S.C | S.D | S.E |
|---|---|---|---|---|---|
| a | 1 | 10 | 10 | x | 2 |
| a | 1 | 10 | 20 | y | 2 |
| . | | | | | |
| . | | | | | |
| c | 2 | 10 | 10 | x | 2 |
| . | | | | | |
| . | | | | | |

RXS

| R.A | R.B | R.C | S.C | S.D | S.E |
|---|---|---|---|---|---|
| a | 1 | 10 | 10 | x | 2 |
| a | 1 | 10 | 20 | y | 2 |
| . | | | | | |
| . | | | | | |
| c | 2 | 10 | 10 | x | 2 |
| . | | | | | |
| . | | | | | |

元组匹配!

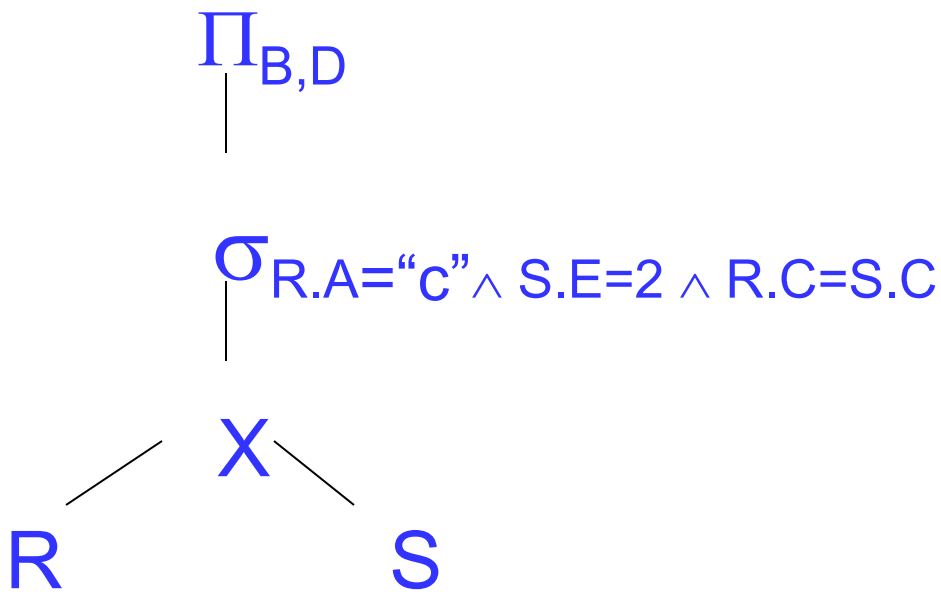
一条结果...

Select B,D From R,S

Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C

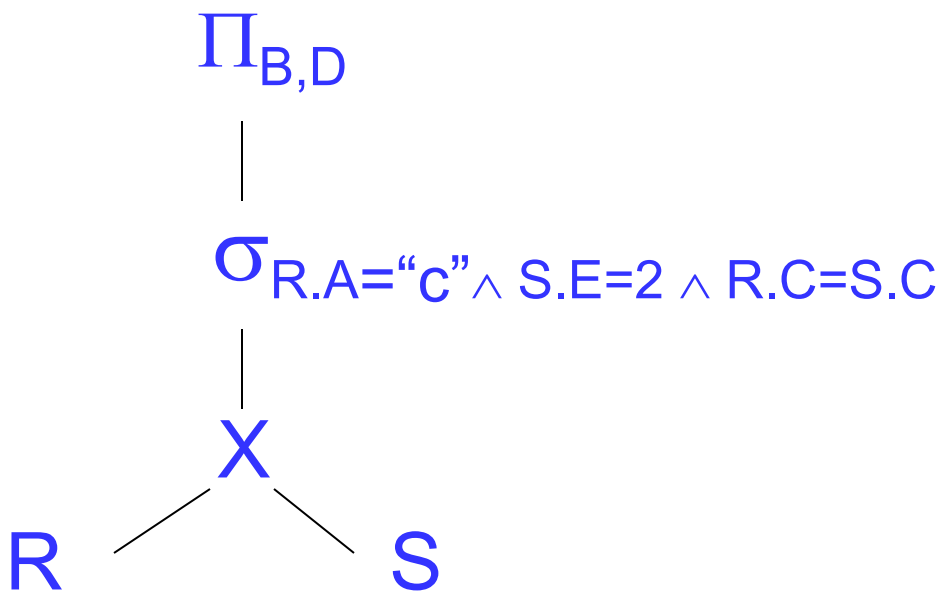
# 关系代数 - 用来描述查询计划...

查询计划 I



# 关系代数 - 用来描述查询计划...

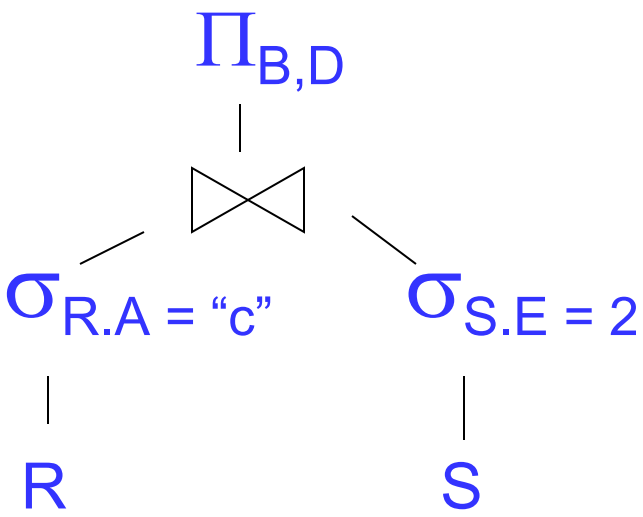
查询计划 I

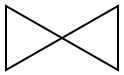



OR: $\Pi_{B,D}\,[\,\sigma_{R.A=\text{“c”} \wedge S.E=2 \wedge R.C = S.C}\,(RXS)]$

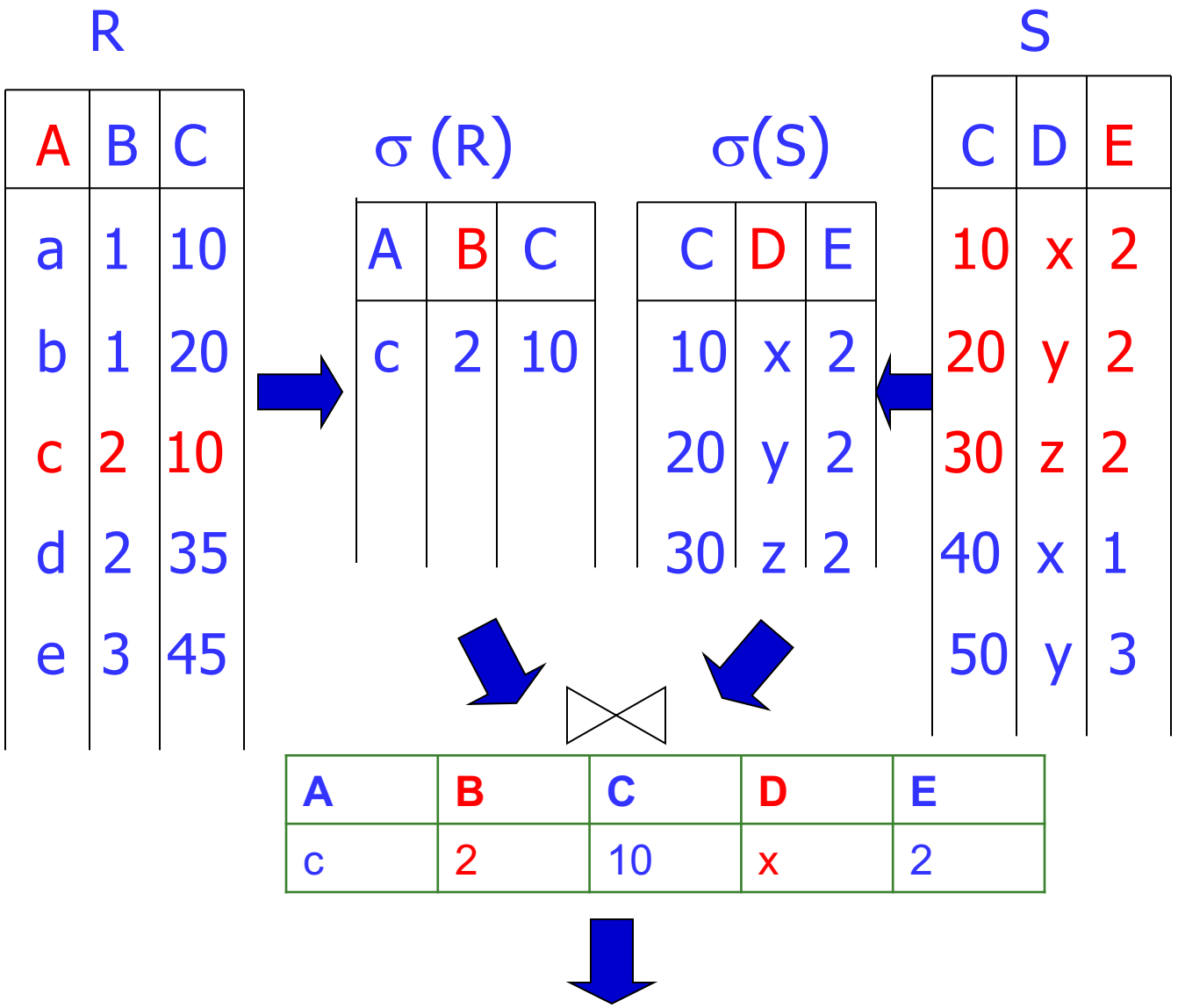
# 其他方法:

## 查询计划 II



natural join


R
A B C
a 1 10
b 1 20
c 2 10
d 2 35
e 3 45
σ (R)
A B C
c 2 10
σ(S)
C D E
10 x 2
20 y 2
30 z 2
S
C D E
10 x 2
20 y 2
30 z 2
40 x 1
50 y 3
A B C D E
c 2 10 x 2

# 查询计划 III

用R.A 和 S.C 上的索引

(1) 首先利用R.A上的索引选择R中的元组：满足R.A = “c”

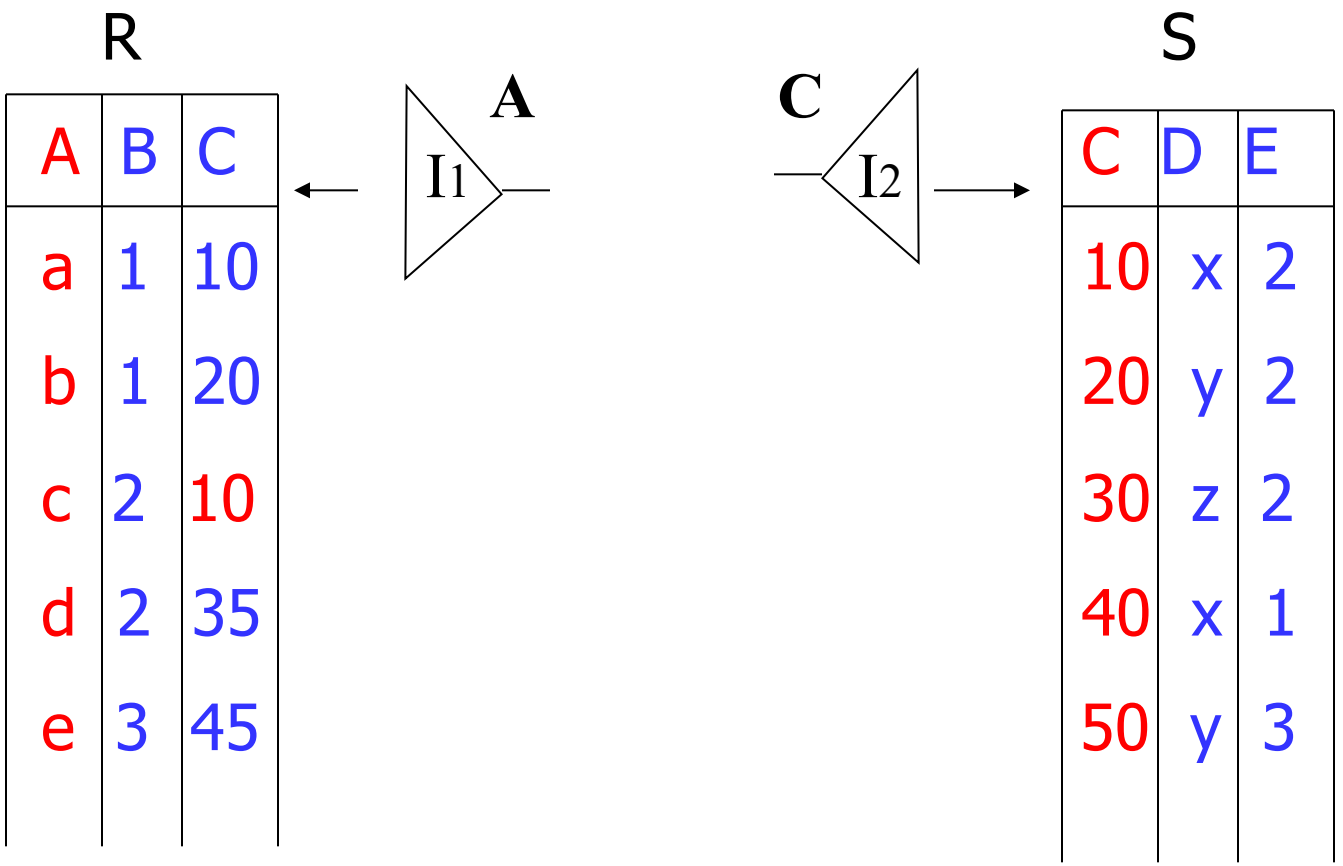
(2) 根据选中的元组的R.C值，利用S.C上的索引选择S上匹配的元组

(3) 去除S中S.E ≠ 2的元组

(4) 将R和S匹配的元组连接, 并在B,D 属性上做投影，获得最终结果

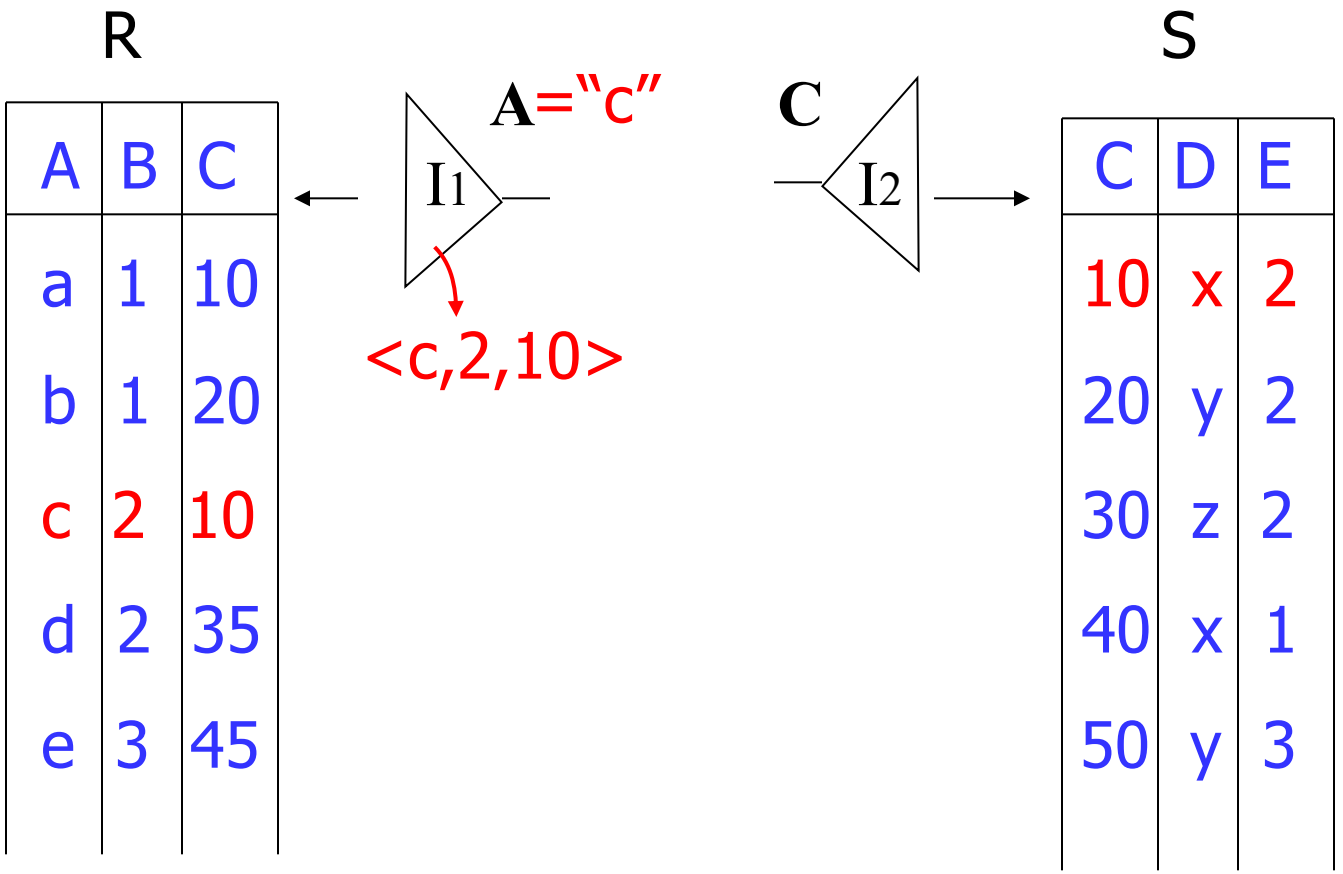
Select B,D From R,S

Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C



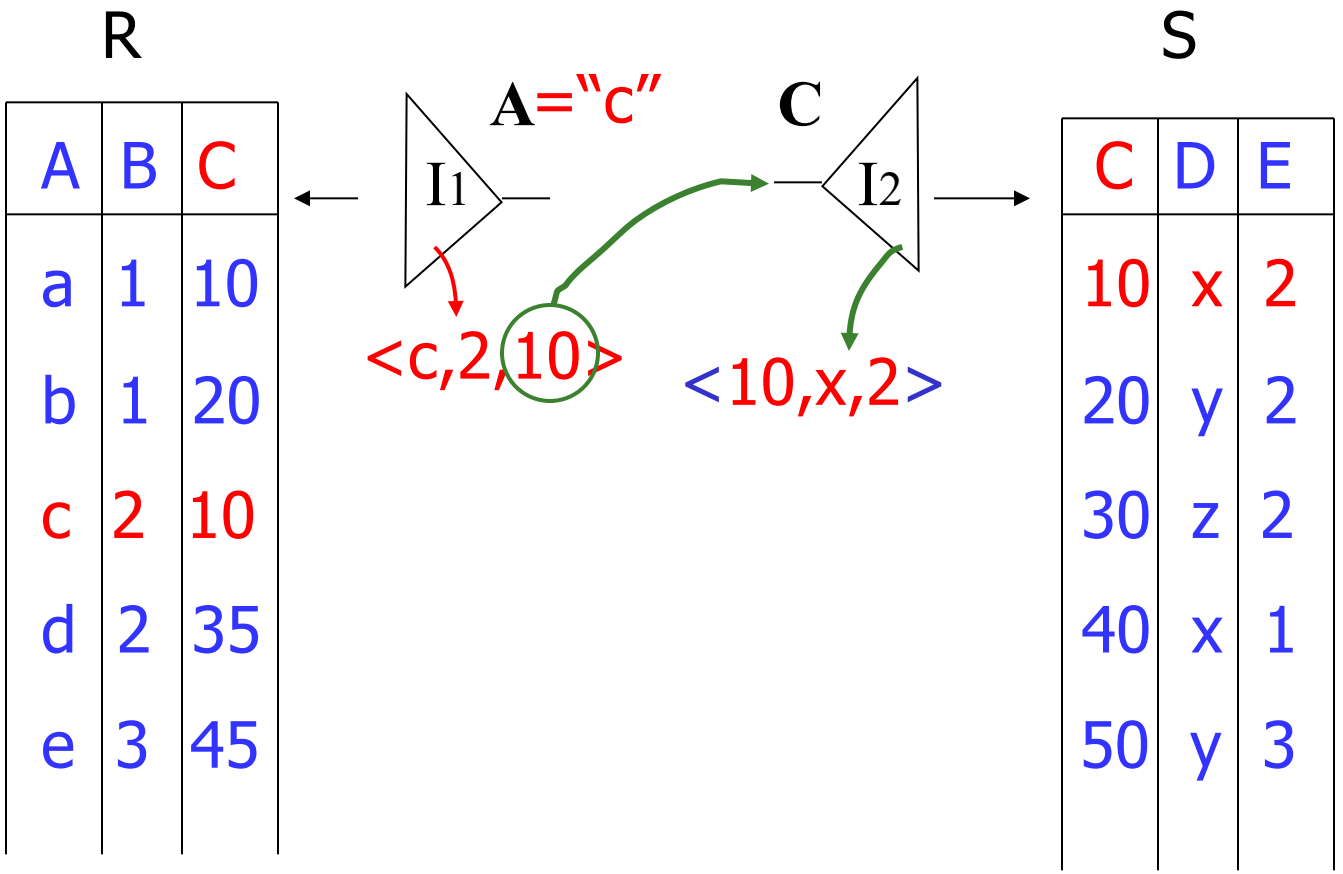

Select B,D From R,S

Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C



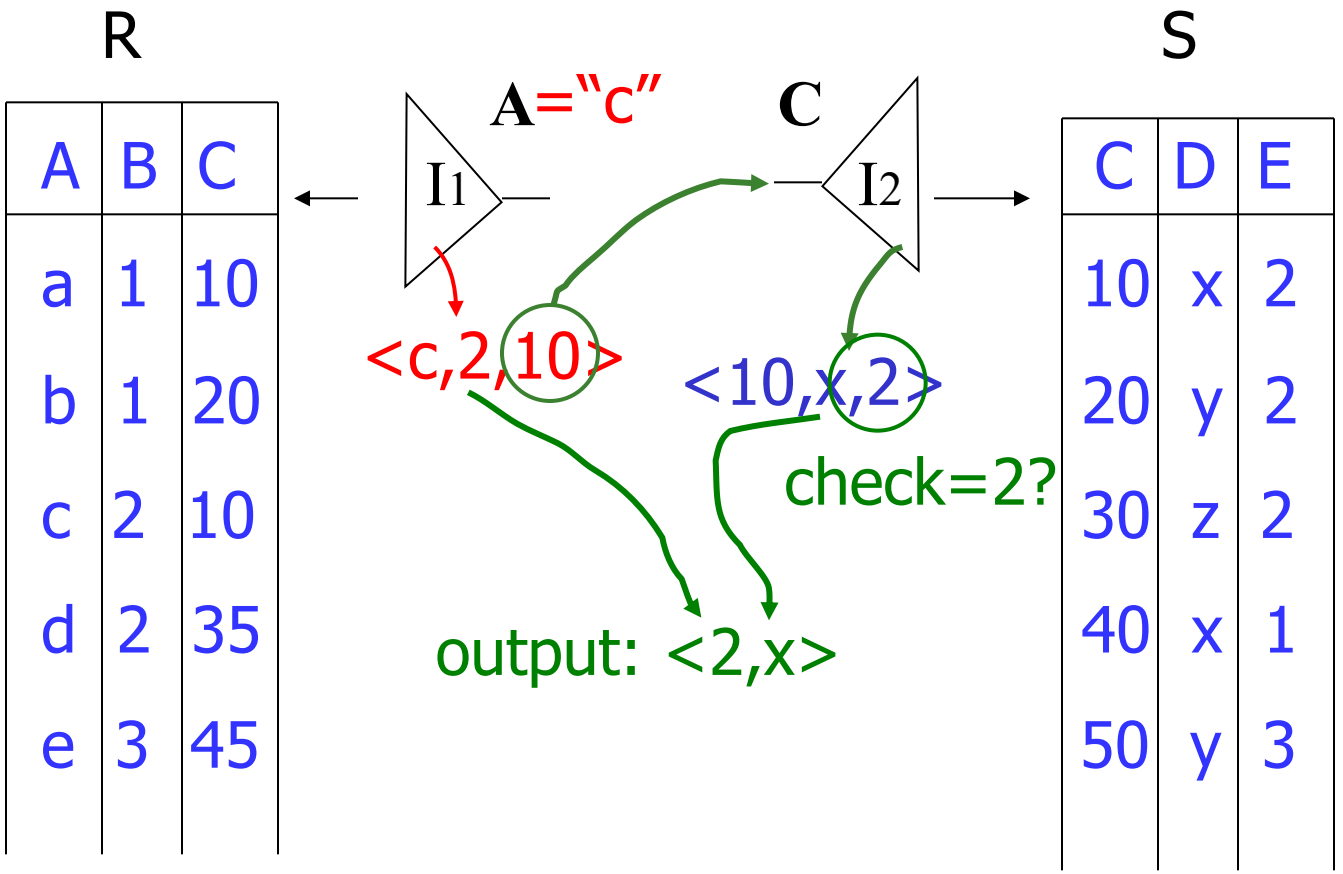

Select B,D From R,S

Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C




Select B,D From R,S

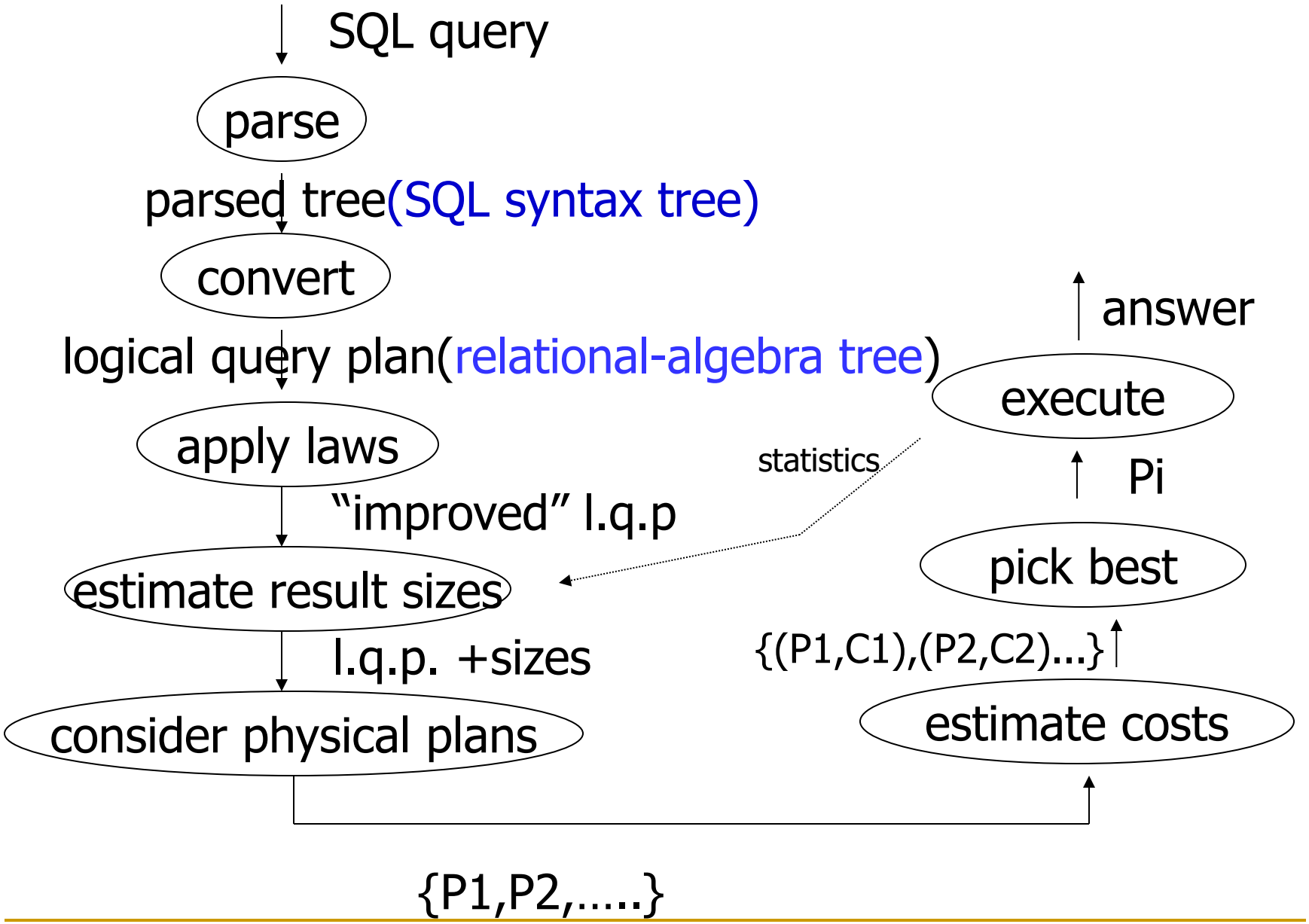
Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C




Select B,D From R,S

Where R.A = “c” ∧ S.E = 2 ∧ R.C=S.C

# 查询优化全景概览


SQL query
parse
parsed tree(SQL syntax tree)
convert
logical query plan(relational-algebra tree)
apply laws
"improved" l.q.p
estimate result sizes
l.q.p. +sizes
consider physical plans
{P1,P2,.....}
answer
execute
Pi
pick best
{(P1,C1),(P2,C2)...}
estimate costs
statistics

两个关系

starsin (title, year, starname)

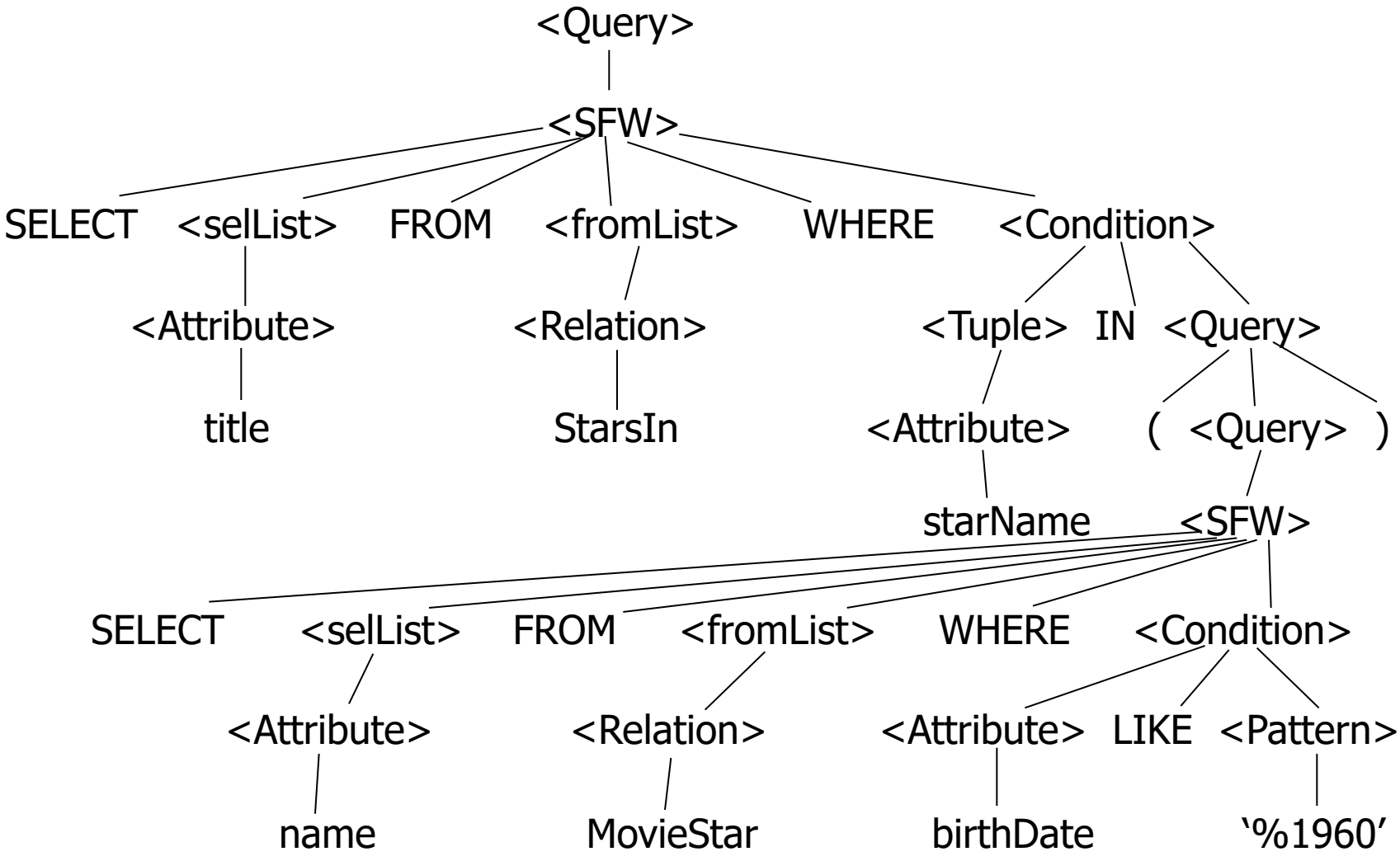
moviestar(name,address,gender,birthday）

# 例子：SQL query

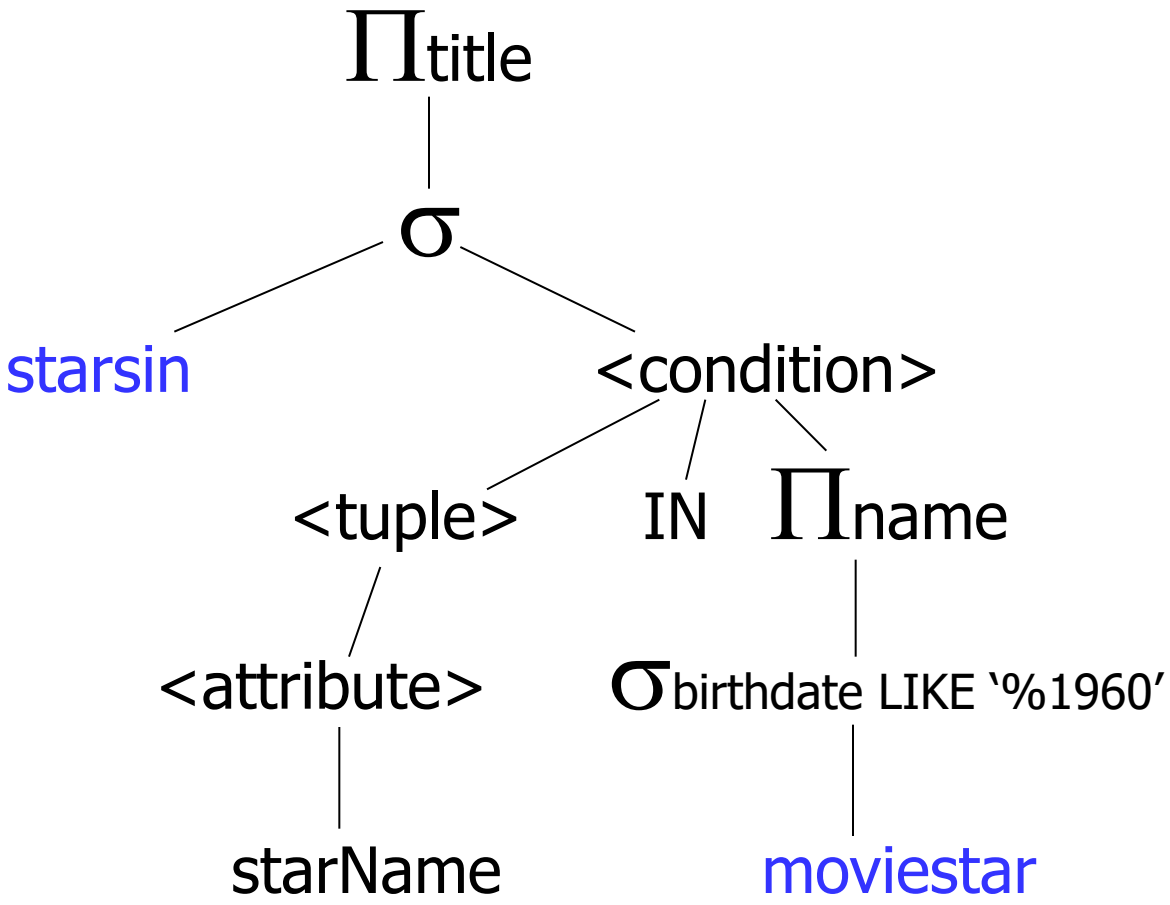
```
SELECT title
FROM starsin
WHERE starname IN
(
        SELECT name
        FROM moviestar
        WHERE birthdate LIKE '%1960'
);
```

列出演员中有1960年出生的演员的电影

# 例子： Parsed Tree



# 例子:关系代数表达式树,tree of relational algebra (logical query plan)



# 例子： 初始逻辑查询计划（7.3.2）

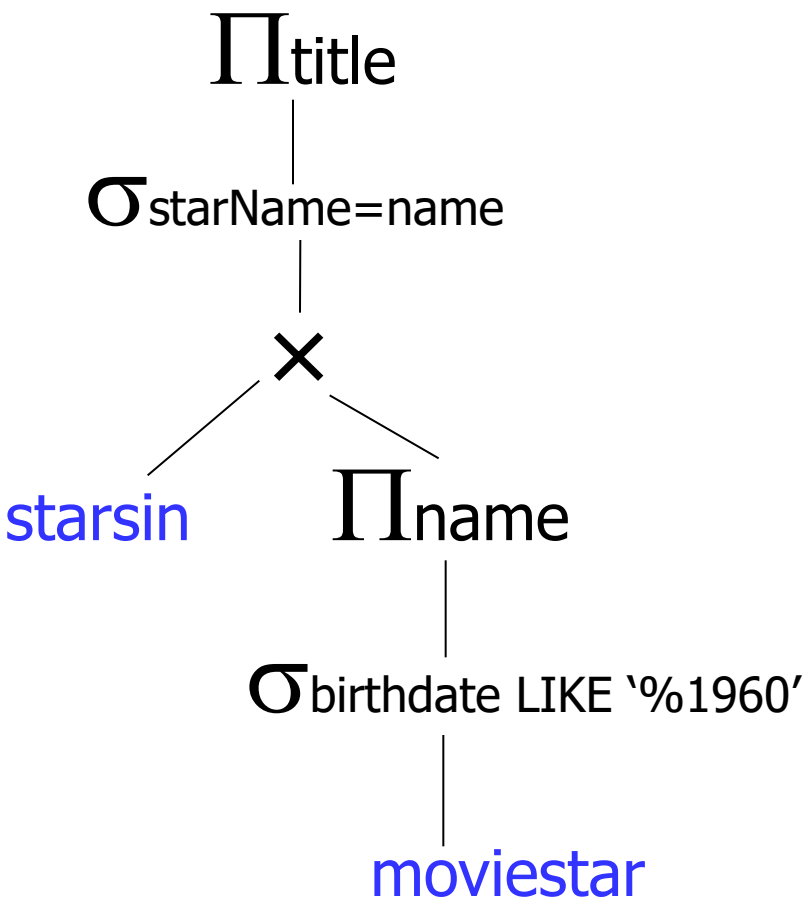
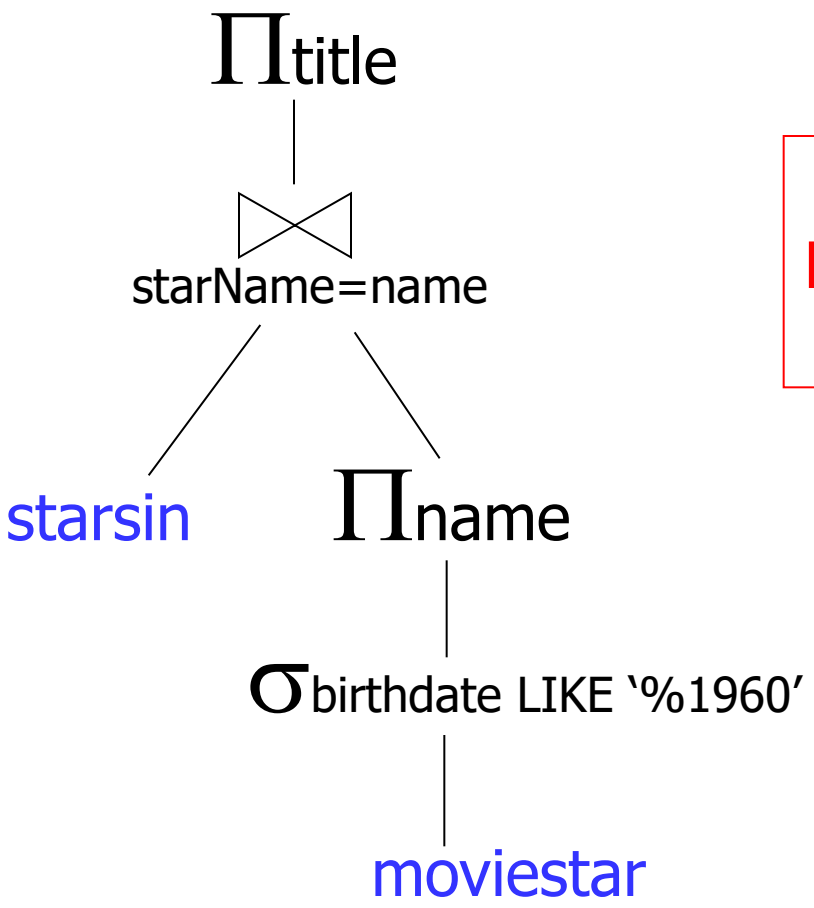



Fig. 7.18: Applying the rule for IN conditions

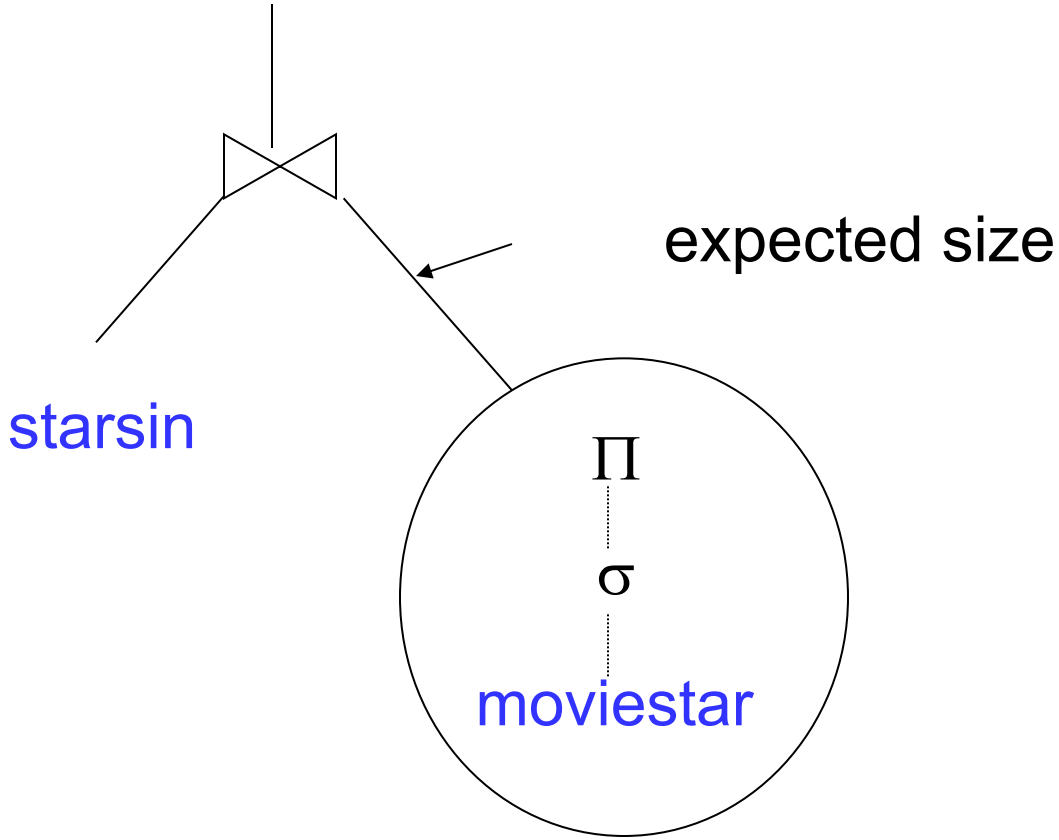
# 例子： 改进的逻辑查询计划



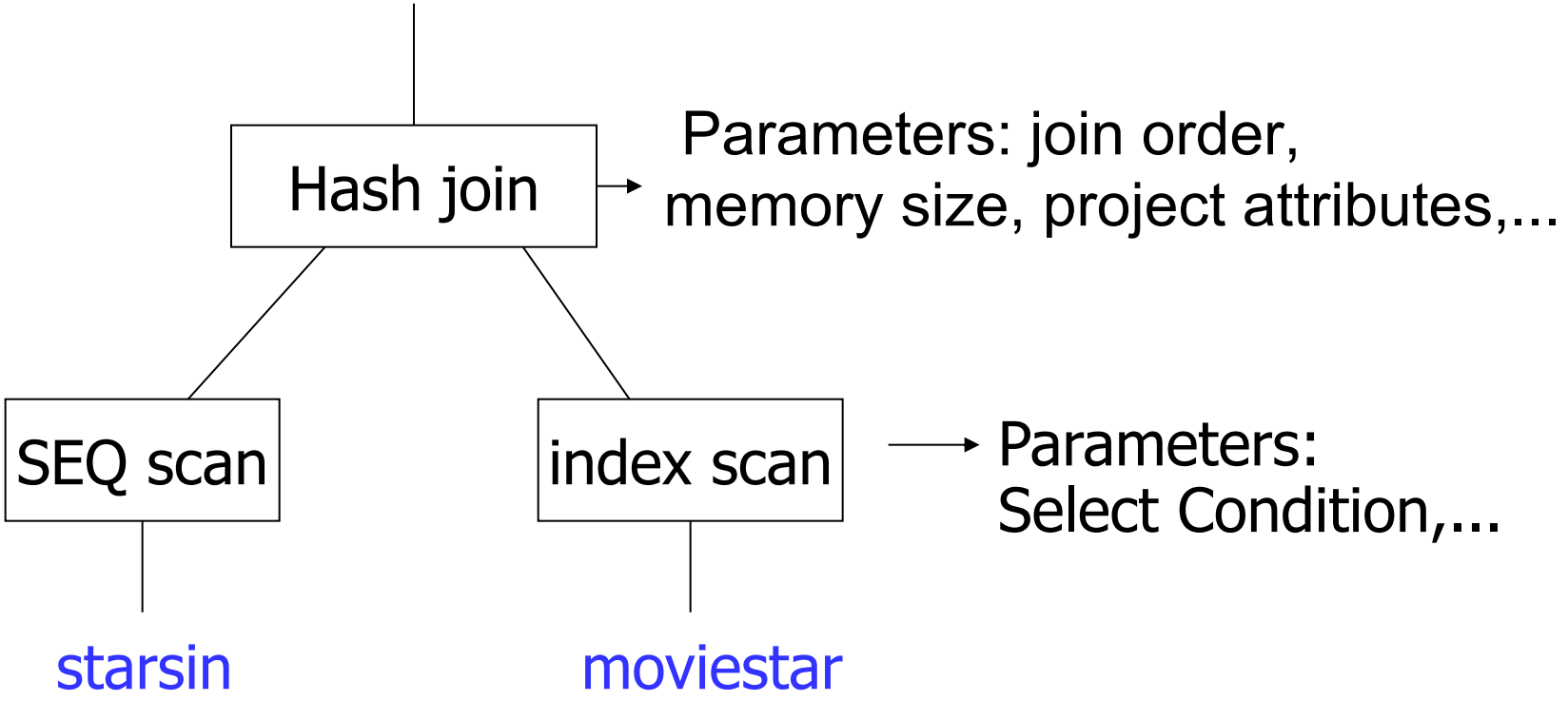

Question:
Push project to StarsIn?

Fig. 7.22: An improvement

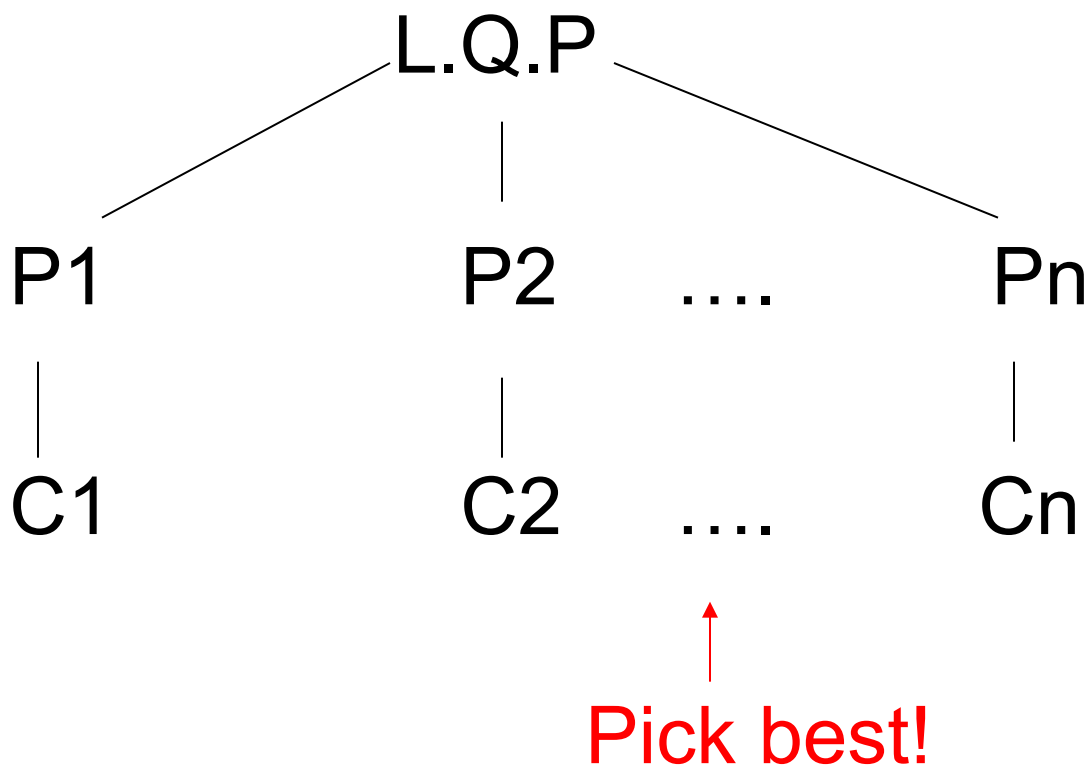
# 例子： 估计结果（包含中间结果）的大小



# 例子: 一个物理执行计划


Hash join
Parameters: join order,
memory size, project attributes,...
SEQ scan
index scan
Parameters:
Select Condition,...
starsin
moviestar

# 例子: 代价估计



# 内容提要

1 语法分析

2 逻辑查询计划（代数优化）

3 结果大小估计（代价优化的基础）- 选讲

# 语法分析

1、语法分析器接受SQL语句，并把它转换为语法分析树

2、语法分析树的结点对应以下两者之一

（1）原子（叶子结点）

它们是词法成分，如关键字（如select、from、 where、 in、like等）、关系名、属性名、常数、括号、操作符(如<,+)等

（2）语法类（中间结点）

即在一个查询中起相似作用的查询子成分所形成的族的名称。用尖括号将描述性的名称括起来表示语法类。例如<SFW>表示select-from-where,<condition>表示条件

提示：语法分析程序参见编译原理内容。

# SQL的简单查询语句（巴克斯范式，BNF）

1、查询

<Query>::=<SFW>

<Query>::=(<SFW>)

2、select-from-where的形式

<SFW>::=select<selList> from <fromList> where <condition>

3、select列表

<selList>::=<Attribute>,<selList>

<selList>::=<Attribute>

# SQL的语法分析子集

4、from

<fromList>::=<relation>,<fromList>

<fromList>::=<relation>

5、条件

<condition>::=<condition> and <condition>

<condition>::=<tuple> in (<Query>)

<condition>::=<attribute> = <attribute>

<condition>::=<attribute> like <pattern>

<tuple>::=<attribute>

# SQL的语法分析子集

6、基本语法类

<relation>，<attribute>，<pattern>通过其所代表的原子来定义的

提示：原子要通过具体的的数据库确定

# 语法树例子

两个关系

starsin(title,year,starname)

moviestar(name,address,gender,birthday)

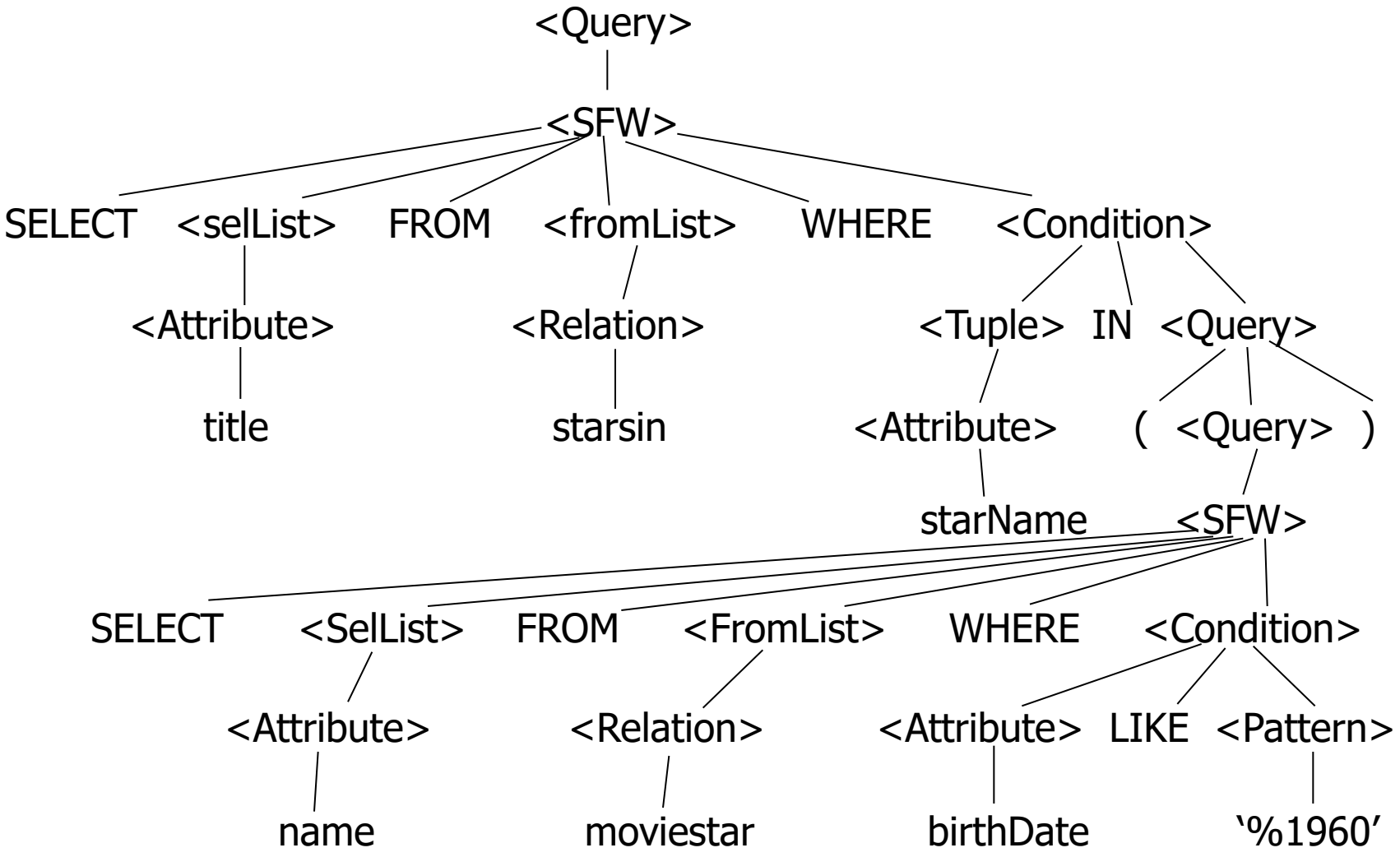
查询：至少有一个在1960年出生的影星的电影的名字

# 语法树例子

```
SELECT title
FROM starsin
WHERE starName IN (
        SELECT name
        FROM moviestar
        WHERE birthdate LIKE '%1960'
);
```

(列出演员中有1960年出生的演员的电影)

# 解析的语法树


<Query>
<SFW>
SELECT
<selList>
FROM
<fromList>
WHERE
<Condition>
<Attribute>
<Relation>
<Tuple>
IN
<Query>
title
starsin
<Attribute>
(
<Query>
)
starName
<SFW>
SELECT
<SelList>
FROM
<FromList>
WHERE
<Condition>
<Attribute>
<Relation>
<Attribute>
LIKE
<Pattern>
name
moviestar
birthDate
'%1960'

# 参见程序

lex_db.py ：

词法分析，识别SQL语句中各个符号（ token）

parser_db.py :

语法分析，形成语法树

query_plan_db.py :

construct_logical_tree()　语法树转换为逻辑查询计划

execute_logical_tree()　执行处理

main_db.py:

第5个执行分支

# lex_db.py

复习编译原理中词法分析器lex的用法

功能：

输入select...from...where语句，识别对应的tokens

定义的tokens：

tokens=('SELECT','FROM','WHERE','AND','TCNAME','EQX','COMMA','CONSTANT','SPACE')

# lex_db.py

SQL例句：

select f1,f2 from GOOD where f1='xx' and f2=5

LexToken(SELECT,'select',1,0)
LexToken(TCNAME,'f1',1,7)
LexToken(COMMA,',',1,9)
LexToken(TCNAME,'f2',1,10)
LexToken(FROM,'from',1,13)
LexToken(TCNAME,'GOOD',1,18)
LexToken(WHERE,'where',1,23)
LexToken(TCNAME,'f1',1,29)
LexToken(EQX,'=',1,31)...

# parser_db.py

学习yacc（Yet Another Compiler Compiler）的用法

功能：

为select...from...where语句创建一棵语法树

原理：

（1）逻辑上通过巴克斯范式递归定义语法

（2）每个函数的开头是语法规则，例子：

'SFW : SELECT SelList FROM FromList WHERE Cond'

（3）每个函数根据语法规则解析传入的token，同时创建树节点，用到了common_db.py中的Node类定义

# query_plan_db.py

construct_logical_tree() 语法树转换为查询计划树

execute_logical_tree()  查询执行

备注：

（1）这两个函数在main_db.py的第5个分支调用

（2）查询处理的四个全局变量，即lex解析器、yacc解析器、全局语法树、逻辑查询计划都在common_db.py中定义，在main_db.py中调用。

# 逻辑查询计划

逻辑查询计划生成器的两阶段任务

（1）第一阶段：生成初始的逻辑查询计划

（2）利用代数定律进行优化，对应教材第一版7.2中的内容

# 生成初始逻辑查询计划

# 初始逻辑查询计划

<SFW>查询，用关系代数表达式替换各个成分——select列表，from列表，简单条件，形成关系代数表达式树，它自下而上由以下内容构成

（1）<fromList>中提及的关系的积是以下操作符的参数

（2）选择$\sigma_C$，其中C是<condition>表达式，同时$\sigma_C$又是下面操作符的参数，对应where条件

（3）投影$\pi_L$，其中L是<selList>中的属性列表

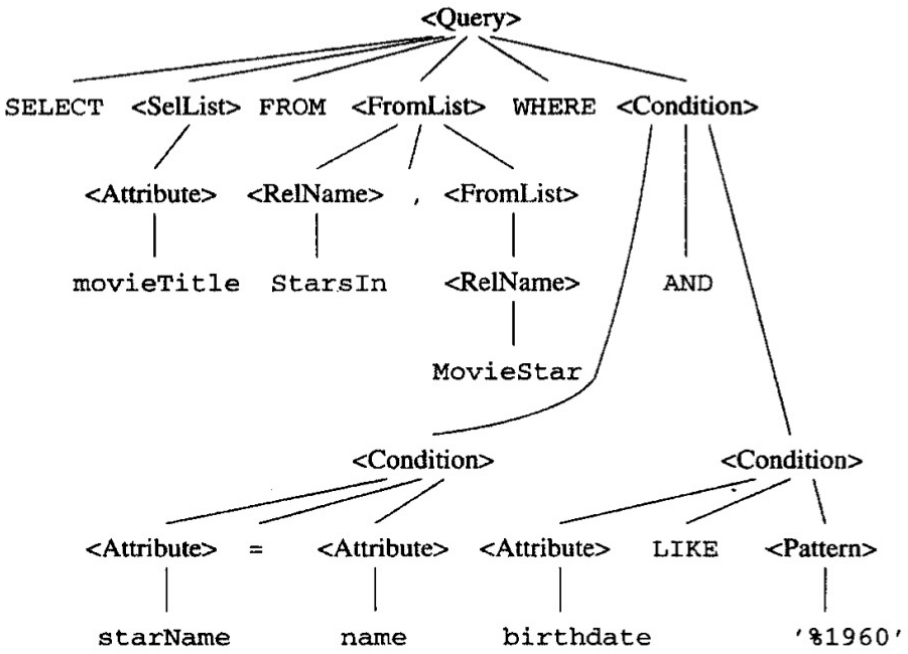
提示：

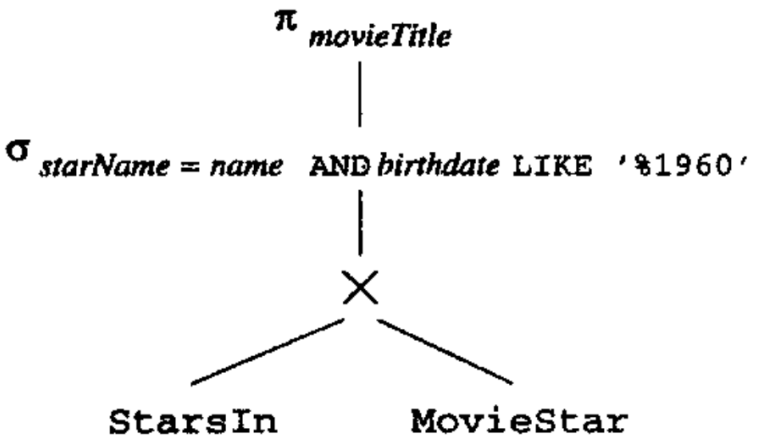
（1）关系代数表达式 =关系代数表达式树

（2）遵循先笛卡尔积，然后选择，然后投影的基本范式

（3）复杂条件参见7.3.2

# 生成初始逻辑查询计划


<Query>
SELECT
<SelList>
FROM
<FromList>
WHERE
<Condition>
<Attribute>
<RelName>
,
<FromList>
movieTitle
StarsIn
<RelName>
AND
MovieStar
<Condition>
<Condition>
<Attribute>
=
<Attribute>
<Attribute>
LIKE
<Pattern>
starName
name
birthdate
'%1960'


π movieTitle
σ starName = name AND birthdate LIKE '%1960'
×
StarsIn
MovieStar

# 逻辑查询计划

逻辑查询计划生成器的两阶段任务

（1）第一阶段：生成初始逻辑查询计划

（2）利用关系代数优化（简称代数优化）

# 代数优化

- 转换规则，即要保持等价性

  两个代数表达式对相同关系，产生相同的属性集合和元组集合，但元组顺序无关紧要

- 什么样的转换好？（经验规则）

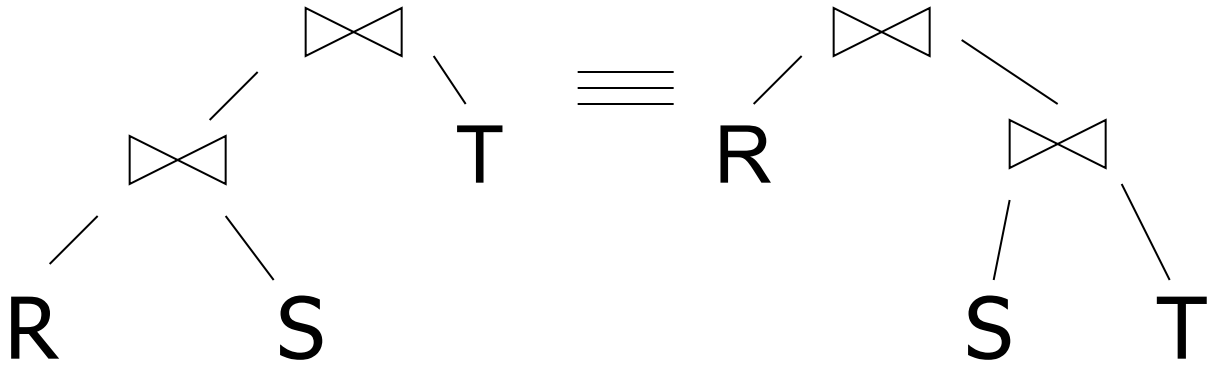
# 转换规则: 自然连接，笛卡尔积，并

$R \bowtie S = S \bowtie R$

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

# 注意:

- 由于各列有属性名，列的顺序不重要
- 也可以用树表达, 例如:



# 转换规则: 自然连接，笛卡尔积，并

$R \bowtie S = S \bowtie R$

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

$R \times S = S \times R$

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

$R \cup S = S \cup R$

$R \cup (S \cup T) = (R \cup S) \cup T$

# 转换规则: 选择

$\sigma_{p1 \wedge p2}(R) =$

$\sigma_{p1 \vee p2}(R) =$

# 转换规则: 选择

$$\sigma_{p1 \wedge p2}(R) = \sigma_{p1}\ [\ \sigma_{p2}\ (R)]$$

$$\sigma_{p1 \vee p2}(R) = [\ \sigma_{p1}\ (R)]\ \cup\ [\ \sigma_{p2}\ (R)]$$

# 规则: 投影

Let: X = 属性集合

Y = 属性集合

XY = X U Y

注意属性舍弃的问题，避免下层把上层还要有的属性提前舍弃了。

# 规则: 选择和连接 （$\sigma + \bowtie$ ）

条件 p： 和R的属性相关联的谓词

q： 和S的属性相关联的谓词

m： 和R与S的属性都关联的谓词

$\sigma_p (R \bowtie S) =$

$\sigma_q (R \bowtie S) =$

# 规则: 选择和连接 （$\sigma + \bowtie$ ）

条件 p： 和R的属性相关联的谓词

q： 和S的属性相关联的谓词

m： 和R与S的属性都关联的谓词

$$\sigma_p (R \bowtie S) = [\sigma_p (R)] \bowtie S$$

$$\sigma_q (R \bowtie S) = R \bowtie [\sigma_q (S)]$$

# 规则：π,σ（投影和选择）结合

条件 x = R属性的子集

z = 谓词P涉及的属性

( R 属性的子集)

$\pi_x[\sigma_{p\,(R)}] =$

# 规则：π,σ（投影和选择）结合

条件 x = R属性的子集

z = 谓词P涉及的属性

( R 属性的子集)

$$\pi_x[\sigma_p(R)] = \{\sigma_p[\pi_x(R)]\}$$

# 规则： π,σ（投影和选择）结合

条件 x = R属性的子集

z = 谓词P涉及的属性

( R 属性的子集)

$$\pi_x[\sigma_p(R)] = \pi_x\{\sigma_p[\pi_{xz}(R)]\}$$

# 规则： $\pi$, $\bowtie$ （投影和连接）结合

Let x = subset of R attributes

y = subset of S attributes

z = intersection of R,S attributes

$\pi_{xy}(R \bowtie S) =$

## 规则: $\pi$, $\bowtie$（投影和连接）结合

Let x = subset of R attributes

y = subset of S attributes

z = intersection of R,S attributes

$$\pi_{xy}(R \bowtie S) =$$

$$\pi_{xy}\{[\pi_{xz}(R)] \bowtie [\pi_{yz}(S)]\}$$

# 常见的“好的”转换?

- $\sigma_{p1 \wedge p2}(R) \rightarrow \sigma_{p1}[\sigma_{p2}(R)]$
- $\sigma_{p}(R \bowtie S) \rightarrow [\sigma_{p}(R)] \bowtie S$
- $R \bowtie S \rightarrow S \bowtie R$
- $\pi_{x}[\sigma_{p}(R)] \rightarrow \pi_{x}\{\sigma_{p}[\pi_{xz}(R)]\}$

# 常见的“好的”转换?（早做投影）

例:

R(A,B,C,D,E)　x={E}

P: (A=3) ∧ (B=“cat”)

$\pi_x\{\sigma_p\ (R)\}$　vs.　$\pi_E\{\sigma_p\{\pi_{ABE}(R)\}\}$

# 总结：关系代数表达式改进的经验原则

（1）选择可尽可能深地下推到表达树中，如果一个选择表达式是多个条件的and,则可以将该条件分解后分别下推每个条件

（2）投影可被下推到树中，或者加入新投影（保留上层的选择属性）

（3）某些选择可以和下面的笛卡尔积结合成等值连接

# 回顾：内容提要

1 语法分析

2 逻辑查询计划生成（代数优化）

3 结果大小估计（代价估算的基础）

提示：查询优化时根据代价估算进行查询计划遍历和选优，代价估算主要考虑I/O代价和内存限制，而这两个受制于结果大小。

# 结果大小估计

- 关系R的统计量
  - T(R) : R中元组的数目
  - S(R) : R中每个元组的长度大小（字节）
  - B(R): 存储R所用的块数目
  - V(R, A) : R中属性A的不同值的数目

# 例子

R

| A | B | C | D |
|---|---|---|---|
| cat | 1 | 10 | a |
| cat | 1 | 20 | b |
| dog | 1 | 30 | a |
| dog | 1 | 40 | c |
| bat | 1 | 50 | d |

A: 20 byte string

B: 4 byte integer

C: 8 byte date

D: 5 byte string

# 例子

R

| A | B | C | D |
|---|---|---|---|
| cat | 1 | 10 | a |
| cat | 1 | 20 | b |
| dog | 1 | 30 | a |
| dog | 1 | 40 | c |
| bat | 1 | 50 | d |

A: 20 byte string

B: 4 byte integer

C: 8 byte date

D: 5 byte string

T(R) = 5

V(R,A) = 3

V(R,B) = 1

S(R) = 37

V(R,C) = 5

V(R,D) = 4

# 乘积大小的估计

# 大小估计　W = R1 x R2

T(W) =

S(W) =

# 大小估计　W = R1 x R2

T(W) = T(R1) × T(R2)

S(W) = S(R1) + S(R2)

# 投影大小的估计

（1）投影后仅元组的长度发生变化，元组的数目不变化

（2）如果投影后要消除重复，要在投影后加一个δ操作符

# 选择大小的估计

# 估计大小 $W = \sigma_{A=a}(R)$

S(W) = S(R)

T(W) = ?

# 例子

R

| A | B | C | D |
|---|---|---|---|
| cat | 1 | 10 | a |
| cat | 1 | 20 | b |
| dog | 1 | 30 | a |
| dog | 1 | 40 | c |
| bat | 1 | 50 | d |

V(R,A)=3

V(R,B)=1

V(R,C)=5

V(R,D)=4

$W = \sigma_{z=val}(R)$ T(W) =

# 例子

| A | B | C | D |
|---|---|---|---|
| cat | 1 | 10 | a |
| cat | 1 | 20 | b |
| dog | 1 | 30 | a |
| dog | 1 | 40 | c |
| bat | 1 | 50 | d |

V(R,A)=3

V(R,B)=1

V(R,C)=5

V(R,D)=4

这个元组被选中的概率是多少?

$W = \sigma_{z=val}(R)$ $T(W) =$

# 例子

| A | B | C | D |
|---|---|---|---|
| cat | 1 | 10 | a |
| cat | 1 | 20 | b |
| dog | 1 | 30 | a |
| dog | 1 | 40 | c |
| bat | 1 | 50 | d |

V(R,A)=3

V(R,B)=1

V(R,C)=5

V(R,D)=4

$W = \sigma_{z=val}(R)$ $T(W) = \frac{T(R)}{V(R,Z)}$

# 前述结论基于的假设1:

Z中的属性在V(R,Z) 中均匀分布（uniformly distributed）

## 假设2:

Z中的属性在DOM(R,Z)的范围内均匀分布

# 例子

R

| A | B | C | D |
|---|---|---|---|
| cat | 1 | 10 | a |
| cat | 1 | 20 | b |
| dog | 1 | 30 | a |
| dog | 1 | 40 | c |
| bat | 1 | 50 | d |

Alternate assumption

V(R,A)=3 DOM(R,A)=10
V(R,B)=1 DOM(R,B)=10
V(R,C)=5 DOM(R,C)=10
V(R,D)=4 DOM(R,D)=10

$W = \sigma_{z=val}(R)$ T(W) = ?

# 例子

R

| A | B | C | D |
|---|---|---|---|
| cat | 1 | 10 | a |
| cat | 1 | 20 | b |
| dog | 1 | 30 | a |
| dog | 1 | 40 | c |
| bat | 1 | 50 | d |

Alternate assumption

V(R,A)=3 DOM(R,A)=10
V(R,B)=1 DOM(R,B)=10
V(R,C)=5 DOM(R,C)=10
V(R,D)=4 DOM(R,D)=10

该元组被选中的概率是多少?

$W = \sigma_{z=val}(R)$ T(W) = ?

# 例子

R

| A | B | C | D |
|---|---|---|---|
| cat | 1 | 10 | a |
| cat | 1 | 20 | b |
| dog | 1 | 30 | a |
| dog | 1 | 40 | c |
| bat | 1 | 50 | d |

Alternate assumption

V(R,A)=3 DOM(R,A)=10
V(R,B)=1 DOM(R,B)=10
V(R,C)=5 DOM(R,C)=10
V(R,D)=4 DOM(R,D)=10

$$W = \sigma_{z=val}(R) \qquad T(W) = \frac{T(R)}{DOM(R,Z)}$$

# 总结：选择基数

SC(R,A) = average # records that satisfy equality condition on R.A

$$SC(R,A) = \begin{cases} \dfrac{T(R)}{V(R,A)} \\ \\ \dfrac{T(R)}{DOM(R,A)} \end{cases}$$

区间选择 $W = \sigma_{z \geq val}(R)$ ?

T(W) = ?

区间选择 $W = \sigma_{z \geq val}(R)$ ?

T(W) = ?

- Solution # 1:

  T(W) = T(R)/2

区间选择 $W = \sigma_{z \geq val}(R)$ ?

T(W) = ?

- Solution # 1:

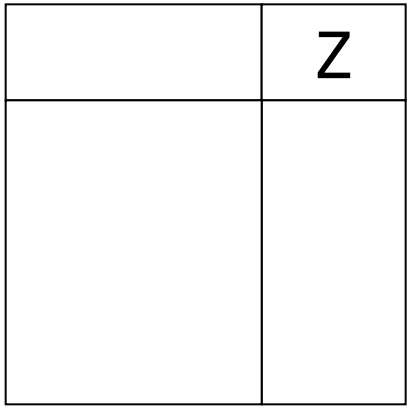
  T(W) = T(R)/2

- Solution # 2:

  T(W) = T(R)/3

- Solution # 3: Estimate values in range

Example R




Min=1

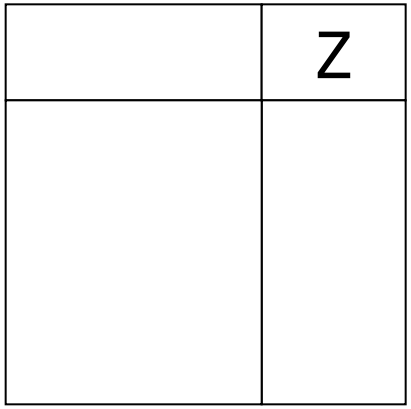
Max=20

V(R,Z)=10

W= $\sigma_{z \geq 15}$ (R)

- Solution # 3: Estimate values in range

Example R




V(R,Z)=10

W= $\sigma_{z \geq 15}$ (R)

$f = \frac{20-15+1}{20-1+1} = \frac{6}{20}$ (区间占的比例)

$T(W) = f \times T(R)$

# 排除性选择连接的基数估计

$S=\sigma_{A\neq'c'}(R)$,其中A是R的属性，c是一个常量。

T(S)?

推荐如下估计：

T(S)=T(R)

# 连接大小的估计(感兴趣的自学)

# 常见的参数获取方法

通过对整个关系进行扫描，DBMS可以计算属性各个值的直方图，常用直方图（用直方图估计结果会更精确）

（1）等宽直方图：选定宽度w(取值范围)及常量$v_0$，提供值为v的元组数计数，v的范围是$v_0 \le v \le v_0+w$, $v_0+w \le v \le v_0+2w$（最常见）

（2）最频值直方图：列出最为公共的值以及他们出现的次数（经常和等宽直方图一起提供）。

# 总结

- 结果大小的估计是一项“艺术”

- 不要忘记:

  统计量必须保持是最新的...

  (当然要有维护代价！)

# 作业

实现一个简单的SQL解析器

补充下列模块的相关代码

（1）main_db.py

（2）lex_db.py

（3）parser_db.py