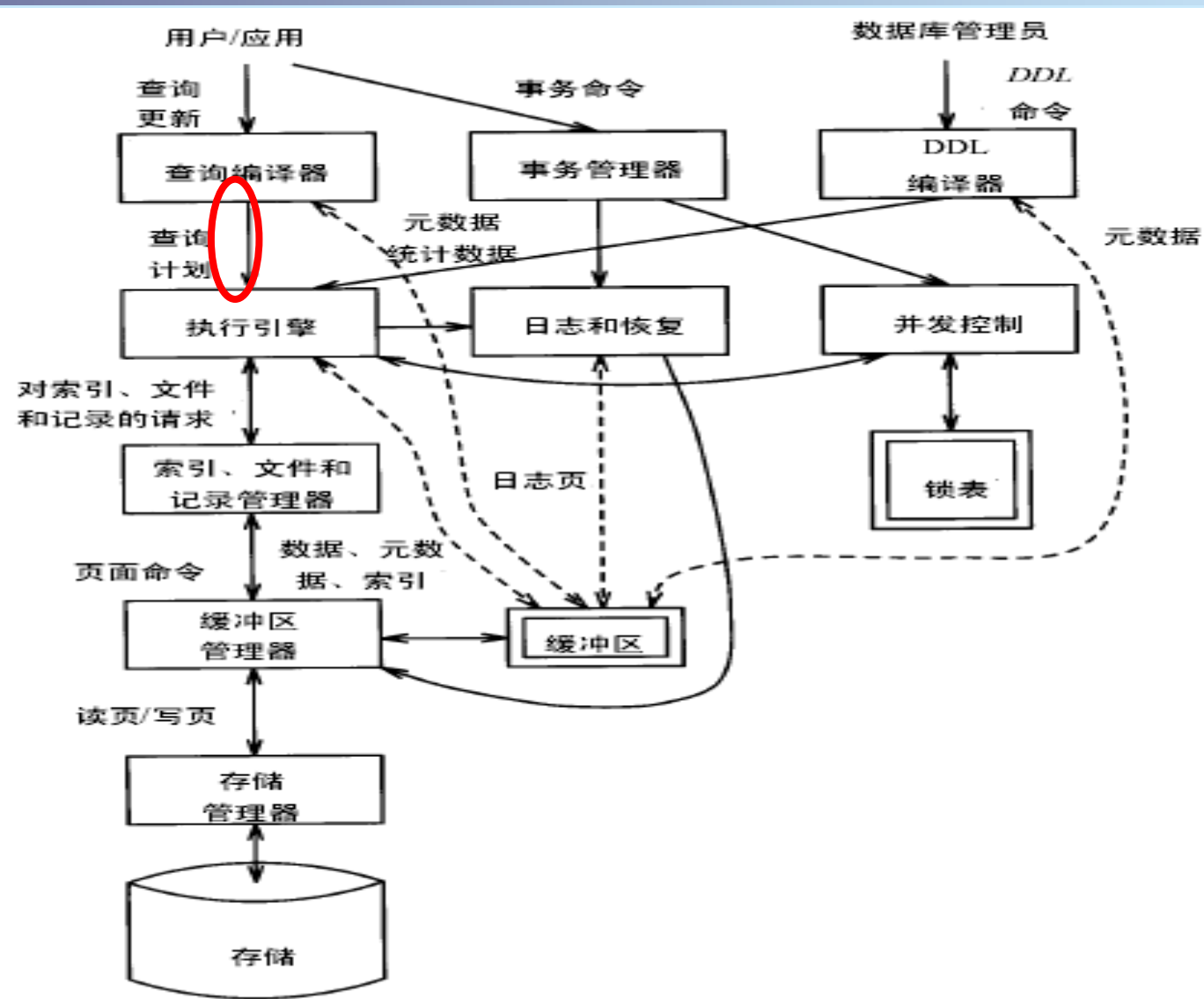


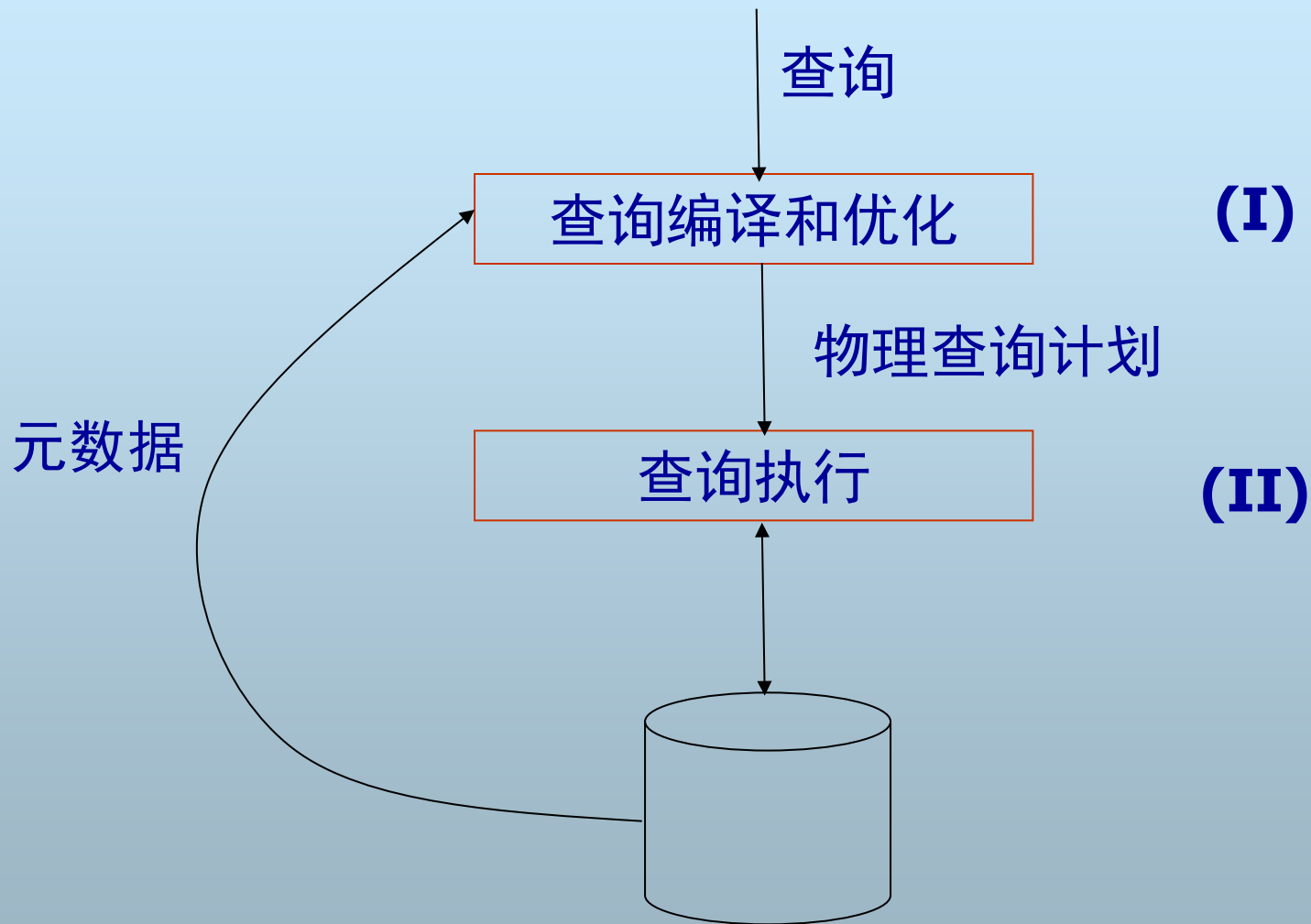
Query Optimization



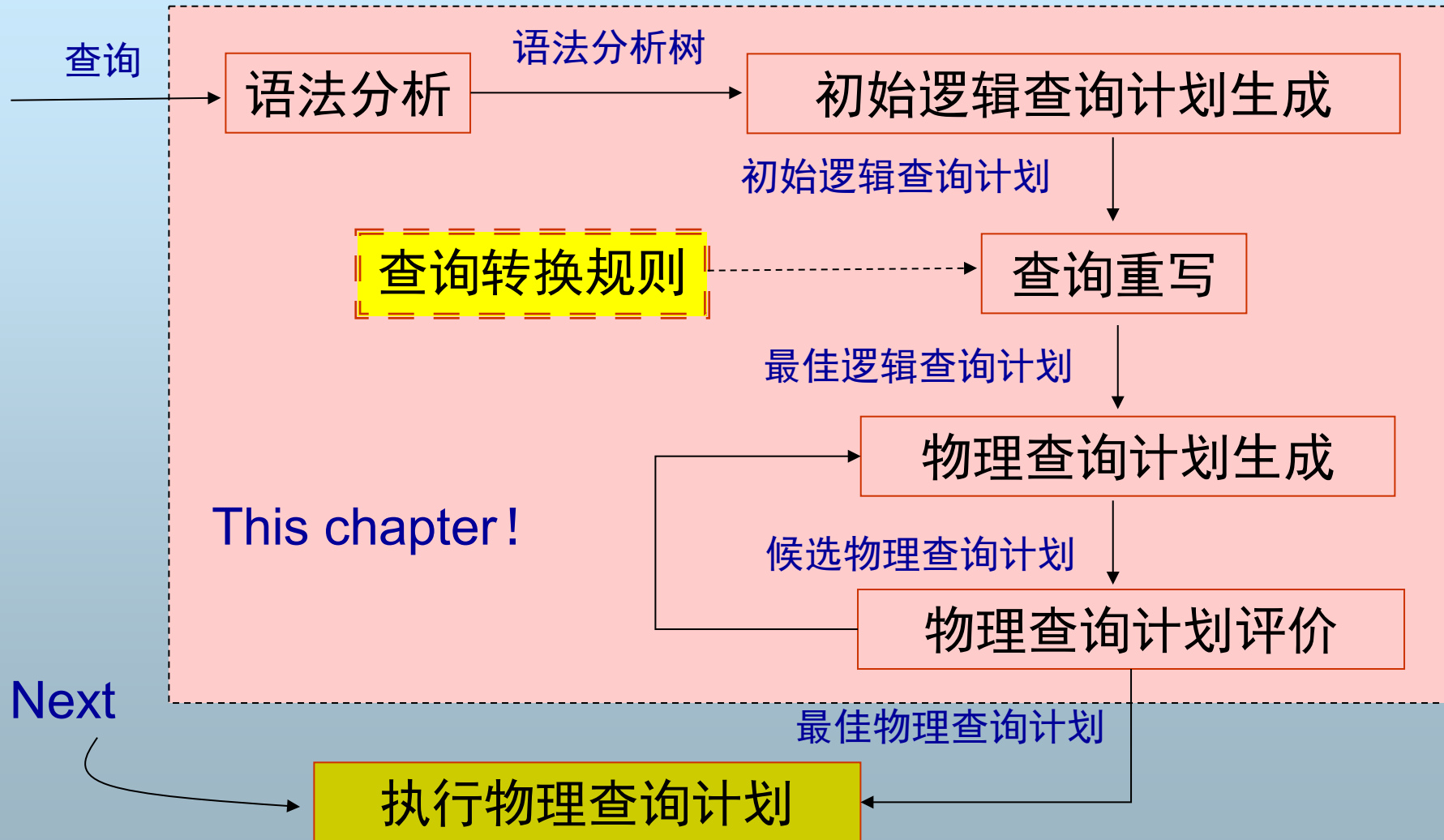
Challenge: How to find out a FAST query plan for a given query, e.g., a SQL query?



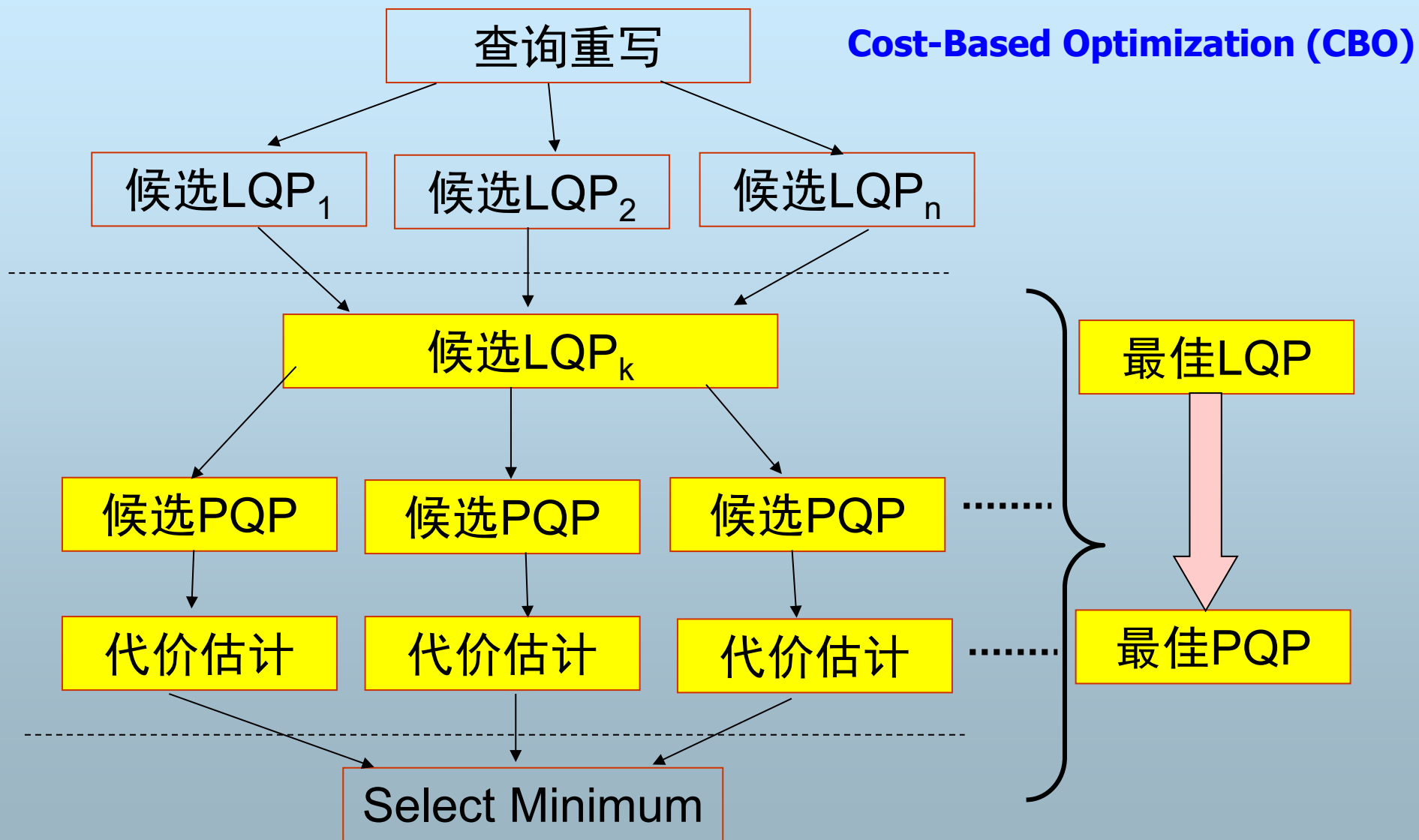
查询处理概述



查询处理概述



查询处理概述



主要内容

- **语法分析(Parsing)**
- **逻辑查询计划生成(Logical Query Plan)**
- **查询重写(Query Rewrite)**
- **查询计划代价估计(Cost Estimation)**
- **物理查询计划选择(Physical Query Plan)**

一、语法分析

- 构造语法分析树(Parsing Tree)

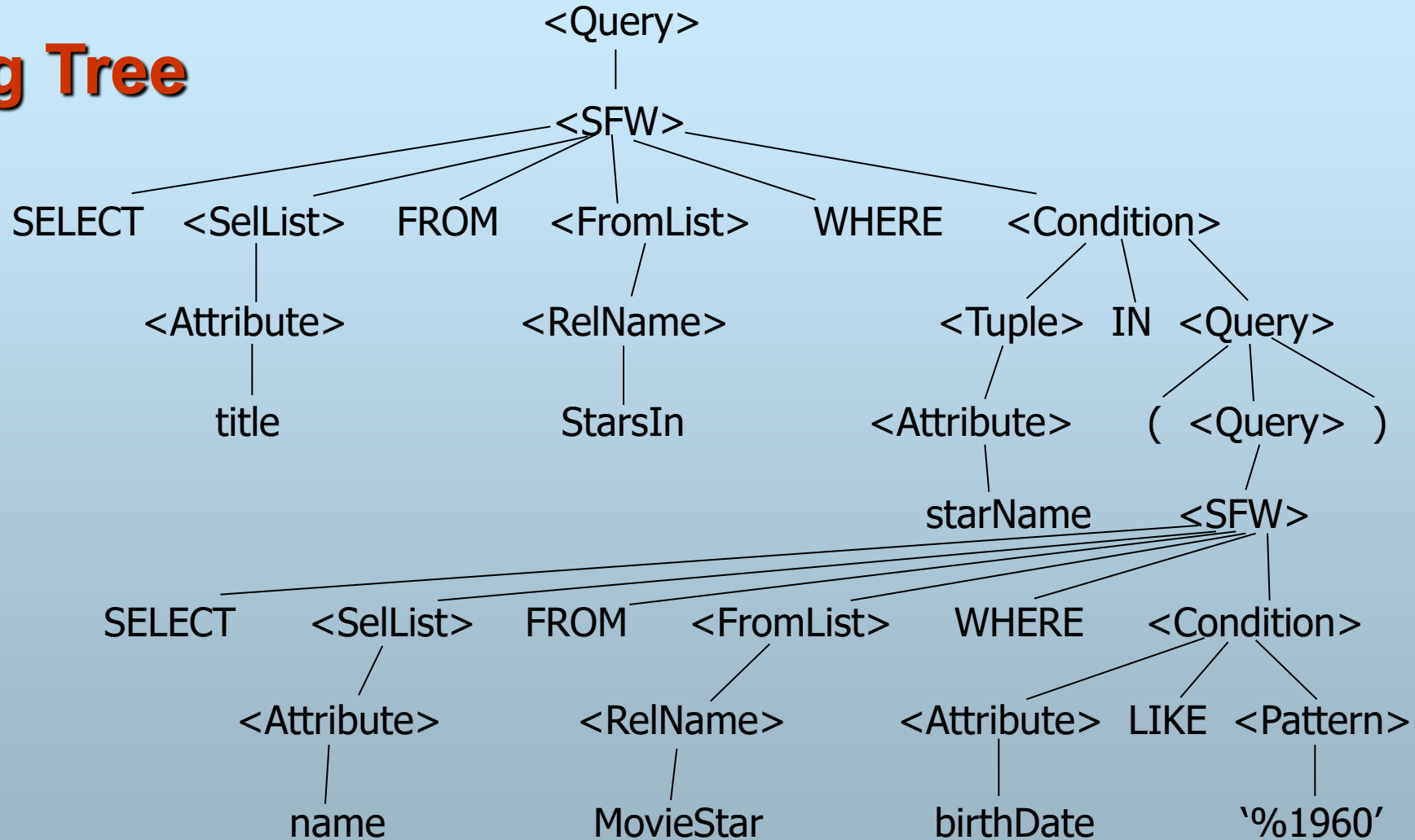
1、SQL查询语法分析

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

(Find the movies with stars born in 1960)

1、SQL查询语法分析

Parsing Tree



二、初始逻辑查询计划生成

- 将语法分析树转换为代数表达式树——逻辑查询计划
 - **Typical:** 关系代数

1、关系代数回顾

- \cup : 并
- \cap : 交
- σ : 选择
- Π : 投影
- $-$: 差
- \bowtie : 自然连接
- \bowtie_{θ} : Theta 连接
-

2、关系代数与SQL

- 关系代数是SQL的代数表达
- 关系代数表达式 \Leftrightarrow SQL查询

■ Note:

- 关系代数基于集合(**SET**)运算
- **SQL**基于包(**BAG**)运算
- 集合：无重复元素
- 包：允许重复元素

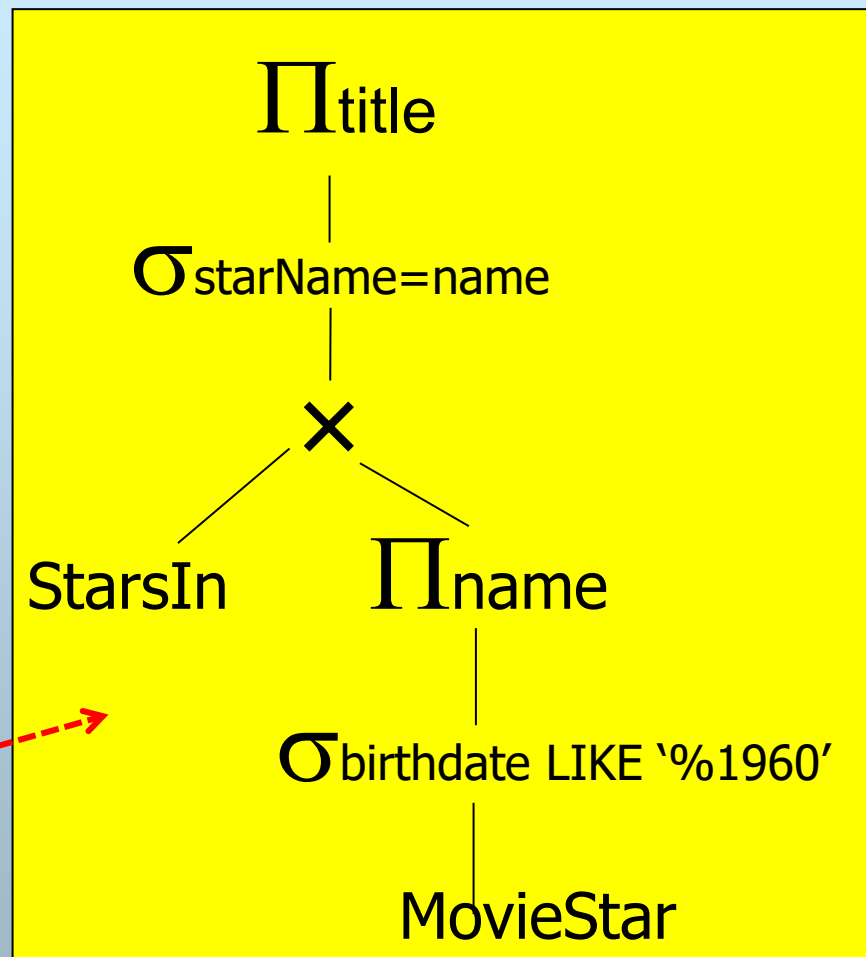
3、逻辑查询计划

- 与语法分析树类似
 - 内结点：关系运算符
 - 叶结点：关系

例：逻辑查询计划生成

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

Logical Query Plan

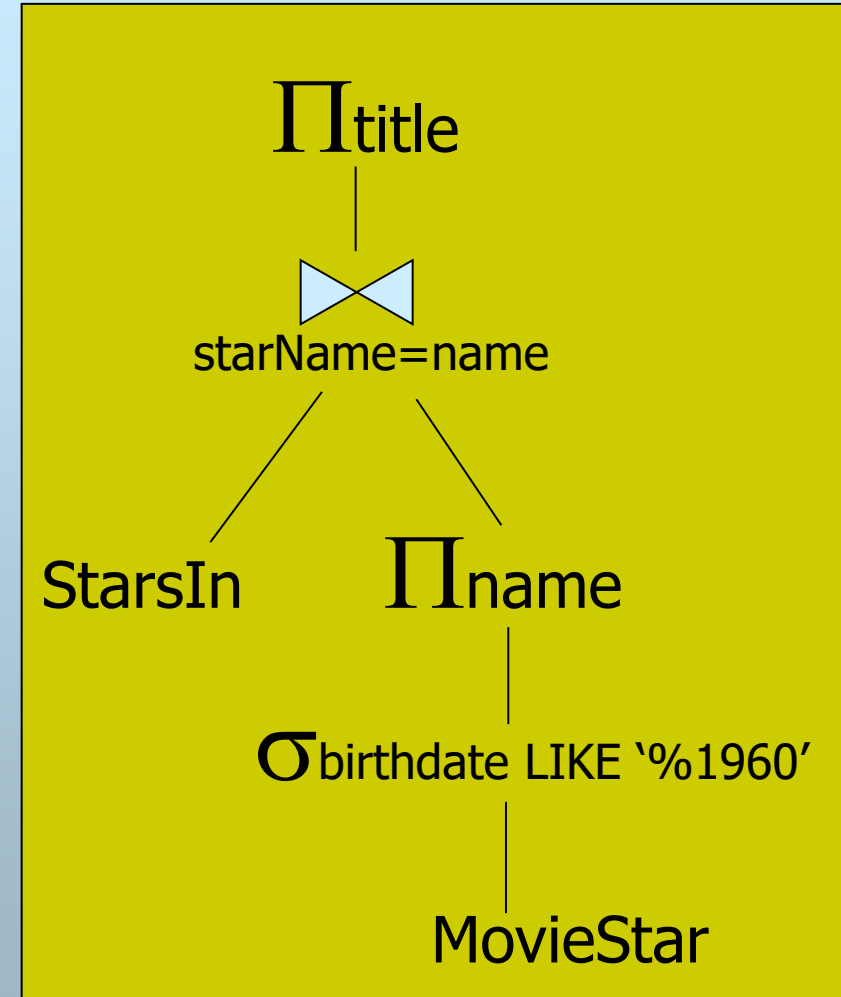
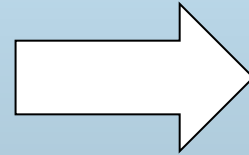
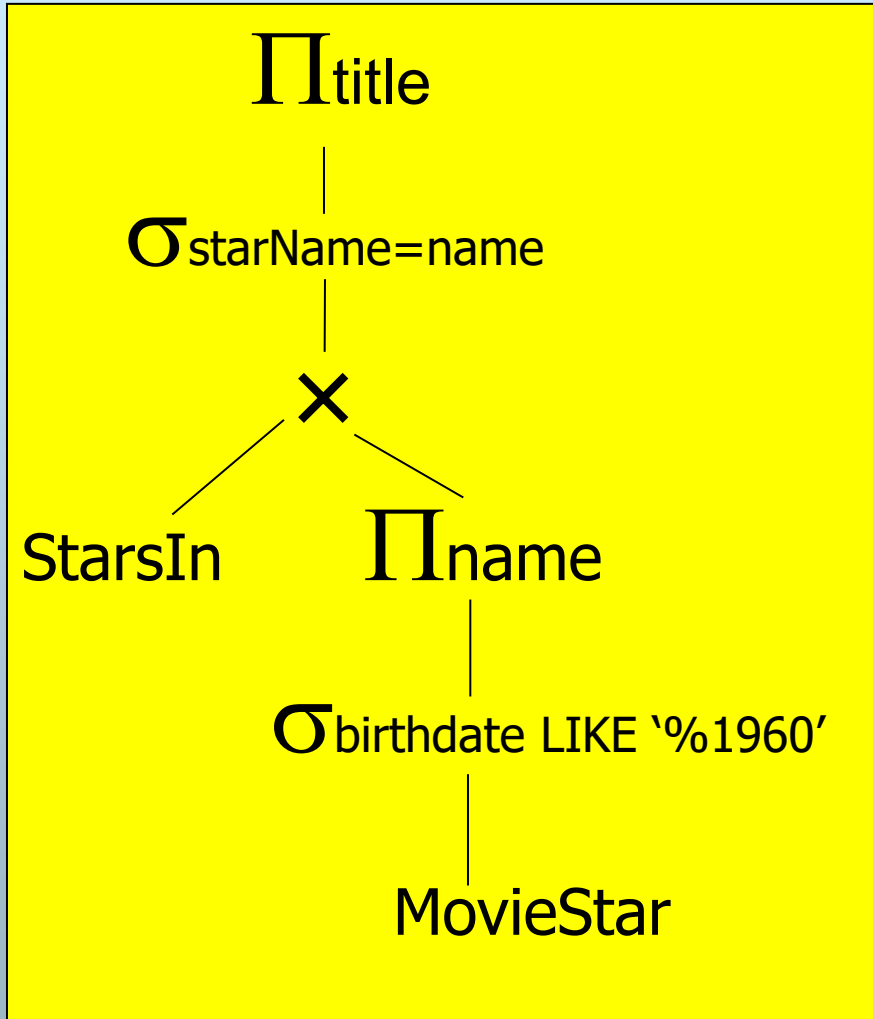


$\Pi_{\text{title}}(\sigma_{\text{starName=name}}(\text{StarsIn} \times \Pi_{\text{name}}(\sigma_{\text{birthdate LIKE '%1960'}}(\text{MovieStar}))))$

三、查询重写

- 将初始逻辑查询计划转换为优化的逻辑查询计划(Maybe)
 - 基于代数转换规则

1、查询重写例子



2、转换规则

- **Transformation rules**
- 运用转换规则，将一个代数表达式转换为另一个等价的代数表达式

2、转换规则

■ 涉及自然连接、并、交、笛卡儿积的交换律和结合律

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

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

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

- $R \cap S = S \cap R; (R \cap S) \cap T = R \cap (S \cap T)$

2、转换规则

R

A	B
10	20
20	30
30	40

S

A	C
10	20
20	30
30	40

T

C	D
20	20
10	30
10	40

- $(R \bowtie S) \bowtie T$: 中间结果 $R \bowtie S$ 产生 3 条记录
- $R \bowtie (S \bowtie T)$: 中间结果 $S \bowtie T$ 产生 1 条记录

查询代价不同

2、转换规则

■ 选择上的转换规则

$$\bullet \sigma_{c_1 \wedge c_2}(R) = \sigma_{c_1}(\sigma_{c_2}(R))$$

$$\bullet \sigma_{c_1 \vee c_2}(R) = (\sigma_{c_1}(R)) \cup_s (\sigma_{c_2}(R))$$

集合并

2、转换规则

■ SQL

- **Union All** — — — 包并
- **Union** — — — 集合并
- 例如，**Student**表和**Staff**表
 - ◆ “返回所有男学生和男教师的姓名”

```
Select name from student where gender='M'  
Union All  
Select name from staff where gender='M'
```

2、转换规则

■ 选择+自然连接

Let p = predicate with only R attribs

q = predicate with only S attribs

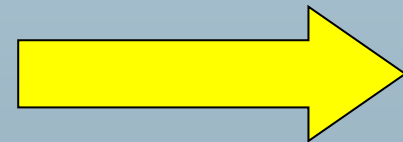
m = predicate with only R,S attribs

下推选择：选择尽可能早做

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

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

可推出另一些规则



2、转换规则

■ 选择+自然连接

$$\sigma_{p \wedge q} (R \bowtie S) = [\sigma_p (R)] \bowtie [\sigma_q (S)]$$

$$\sigma_{p \wedge q \wedge m} (R \bowtie S) = \sigma_m \left[(\sigma_p R) \bowtie (\sigma_q S) \right]$$

$$\sigma_{p \vee q} (R \bowtie S) =$$

$$\left[(\sigma_p R) \bowtie S \right] \cup \left[R \bowtie (\sigma_q S) \right]$$

2、转换规则

■ 投影+自然连接

Let $x =$ subset of R attributes

$y =$ subset of S attributes

$z =$ intersection of R, S attributes

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

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

3、转换规则的几点思考

■ 转换的最终目的

- 减少查询的开销(I/O次数)

■ 转换的直接目的

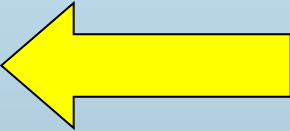
- 减少查询执行时的中间关系大小（元组数）
- 减少元组的大小

Do select early
Choose join orders

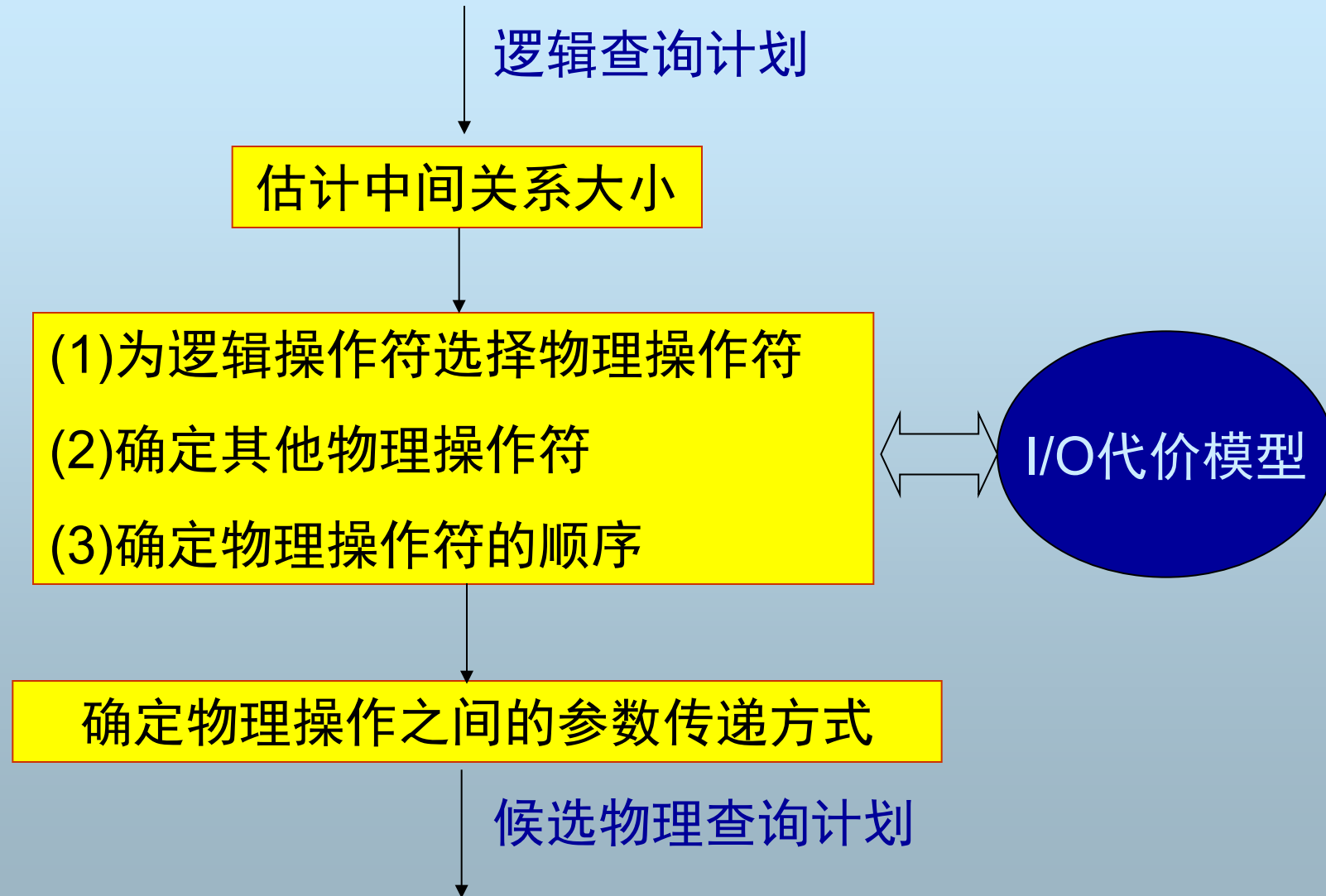
The diagram shows two orange arrows originating from the list items. One arrow points from '减少查询执行时的中间关系大小（元组数）' to the yellow box containing 'Do select early' and 'Choose join orders'. The other arrow points from '减少元组的大小' to the yellow box containing 'Do project before selections'.

Do project before selections

Where are we?

- 语法分析(Parsing)
- 逻辑查询计划生成(Logical Query Plan)
- 查询重写(Query Rewrite)
- 查询计划代价估计  Next
- 物理查询计划选择(Physical Query Plan)

四、查询代价估计(Cost Estimation)



四、查询代价估计(Cost Estimation)

- 中间关系大小估计
- I/O代价估计
- 物理查询计划生成

1、中间结果的大小估计

- 需要使用一些统计量(**statistics**)
 - **$T(R)$** : R 的元组数
 - **$S(R)$** : R 中每个元组的大小(**bytes**)
 - **$V(R, A)$** : R 的属性 A 上的不同值数
 - **$B(R)$** : 容纳 R 所有元组所需的块数
- **These statistics should be held in the database!**

1、中间结果的大小估计

Example

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

$$T(R) = 5$$

$$S(R) = 37$$

$$V(R,A) = 3$$

$$V(R,C) = 5$$

$$V(R,B) = 1$$

$$V(R,D) = 4$$

1、中间结果的大小估计

■ $W = R1 \times R2$ 的大小估计

- $T(W) = T(R1) * T(R2)$
- $S(W) = S(R1) + S(R2)$

1、中间结果的大小估计

- $W = \sigma_{A=a}(R)$ 的大小估计
 - $S(W) = S(R)$
 - $T(W) = ?$

1、中间结果的大小估计

■ $W = \sigma_{A=a}(R)$ 的大小估计

Example

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

$$V(R,A)=3$$

$$V(R,B)=1$$

$$V(R,C)=5$$

$$V(R,D)=4$$

1、中间结果的大小估计

- $W = \sigma_{z=val}(R)$: 假设 z 上的值在 $V(R,z)$ 个不同值上均匀分布

Example

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

$$V(R,A)=3$$

$$V(R,B)=1$$

$$V(R,C)=5$$

$$V(R,D)=4$$



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

1、中间结果的大小估计

■ $W = \sigma_{z > val}(R)$ 的大小估计

● 一种估计

◆ $T(W) = T(R) / 2$

● 另一种估计

◆ $T(W) = T(R) / 3$

● 使用“范围”



1、中间结果的大小估计

- $W = \sigma_{z > val} (R)$ 的大小估计：使用范围

Example

R	Z

Min=1 $V(R,Z)=10$
↕
Max=20 $W = \sigma_{z > 15} (R)$

$$f = \frac{20-15}{20-1+1} = \frac{5}{20} \quad (\text{fraction of range})$$

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

1、中间结果的大小估计

- $W = \sigma_{z \neq \text{val}}(R)$ 的大小估计

$$T(W) = T(R) - \frac{T(R)}{V(R, z)}$$

1、中间结果的大小估计

■ 总结：选择大小的估计 $W = \sigma_p(R)$

■ $T(W) = s * T(R)$

● s 是选中率

$$s = \begin{cases} 1 / V(R,z) & p \text{ 为 “=” 比较时} \\ (V(R,z) - 1) / V(R,z) & p \text{ 为 “\neq” 比较时} \\ \left. \begin{array}{l} 1 / 2 \text{ 或} \\ 1 / 3 \text{ 或} \\ \text{范围命中率 } f \end{array} \right\} & p \text{ 为 “}\geq\text{、}\leq\text{、}\<\text{、}\>\text{” 时} \end{cases}$$

1、中间结果的大小估计

■ $W = R1 \bowtie R2$ 的大小估计

Let $x =$ attributes of $R1$

$y =$ attributes of $R2$

Case 1

$$X \cap Y = \emptyset$$

Same as $R1 \times R2$

1、中间结果的大小估计

Case 2

$$W = R1 \bowtie R2$$

$$X \cap Y = A$$

R1

A	B	C
...

R2

A	D
...	...

Assumption:

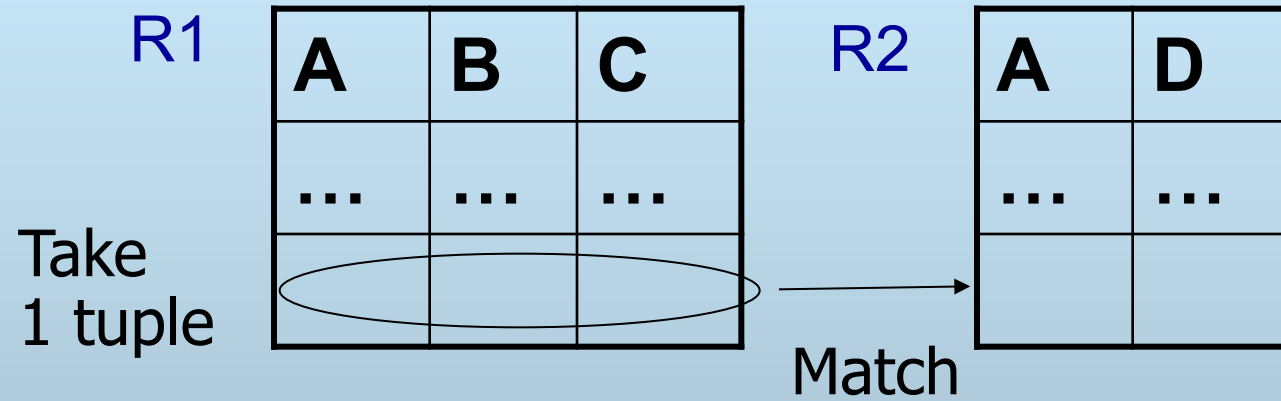
$V(R1,A) \leq V(R2,A) \Rightarrow R1.A$ 上的值都在R2中

$V(R2,A) \leq V(R1,A) \Rightarrow R2.A$ 上的值都在R1中

“值集的包含 containment of value sets”

1、中间结果的大小估计

- $W = R1 \bowtie R2 : V(R1,A) \leq V(R2,A)$



R1中的一个元组在R2中有 $\frac{T(R2)}{V(R2,A)}$ 个元组匹配

$$T(W) = \frac{T(R2)}{V(R2,A)} \times T(R1)$$

1、中间结果的大小估计

- $W = R1 \bowtie R2 : V(R2,A) \leq V(R1,A)$

$$T(W) = \frac{T(R1)}{V(R1, A)} \times T(R2)$$

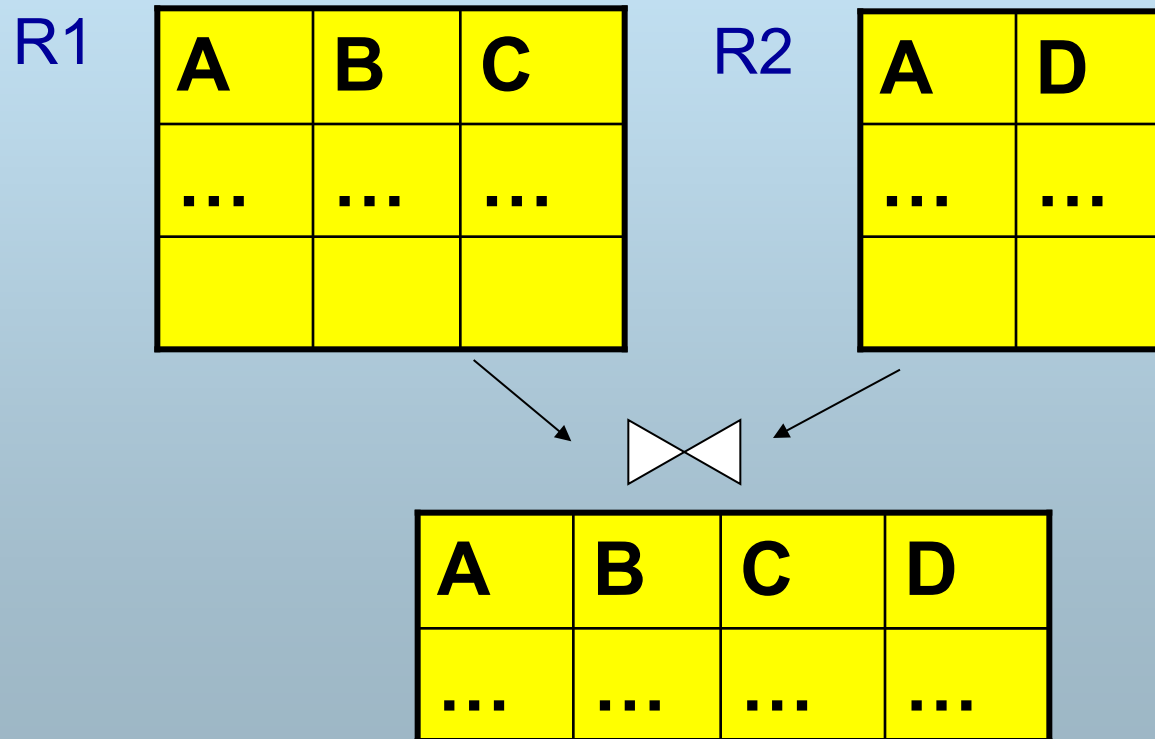
In General

$$T(W) = \frac{T(R1) \cdot T(R2)}{\max \{V(R1, A), V(R2, A)\}}$$

1、中间结果的大小估计

■ $W = R1 \bowtie R2 : S(W) = ?$

● $S(W) = S(R1) + S(R2) - S(A)$



1、中间结果的大小估计

■ $W = R1 \bowtie R2 : V(W,*) = ?$

对于 $W = R1(A,B,C) \bowtie R2(A,D)$

我们可以假设：

$$V(W,B) = V(R1,B)$$

$$V(W,C) = V(R1,C)$$

$$V(W,D) = V(R2,D)$$

$$V(W,A) = \min\{V(R1,A), V(R2,A)\}$$

“preservation of value sets”
值集的保持



假设满足值集的包含

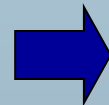
1、中间结果的大小估计

Example

$$Z = R1(A,B) \bowtie R2(B,C) \bowtie R3(C,D)$$

R1	$T(R1) = 1000$	$V(R1,A)=50$	$V(R1,B)=100$
R2	$T(R2) = 2000$	$V(R2,B)=200$	$V(R2,C)=300$
R3	$T(R3) = 3000$	$V(R3,C)=90$	$V(R3,D)=500$

$$U = R1 \bowtie R2$$



$$T(U) = \frac{T(R1) \cdot T(R2)}{\max\{V(R1,B), V(R2,B)\}} = \frac{1000 \times 2000}{200}$$

$$V(U,A) = V(R1,A) = 50$$

$$V(U,B) = \min\{V(R1,B), V(R2,B)\} = 100$$

$$V(U,C) = 300$$

1、中间结果的大小估计

Example(continue)

$$Z = U(A,B,C) \bowtie R3(C,D)$$

U	$T(U) = 10000$	$V(U,A)=50$	$V(U,B)=100$	$V(U,C)=300$
R3	$T(R3) = 3000$	$V(R3,C)=90$	$V(R3,D)=500$	

$$Z = U \bowtie R3$$

$$T(Z) = \frac{T(U) \cdot T(R3)}{\max\{V(U,C), V(R3,C)\}} = \frac{10000 \times 3000}{300}$$

$$V(Z,A) = V(U,A) = 50$$

$$V(Z,B) = V(U,B) = 100$$

$$V(Z,C) = \min\{V(U,C), V(R3,C)\} = 90$$

$$V(Z,D) = V(R3,D) = 500$$

1、中间结果的大小估计

■ 其它情况的代价估计

● $\Pi_{AB}(R)$: see Sec. 16.4.2

● $\sigma_{A=a \wedge B=b}(R)$: see Sec. 16.4.3

● $R \bowtie S$ with multiple common attribs. : see Sec. 16.4.5

● Union, intersection, diff, :
see Sec. 16.4.7

Where are we?

- 语法分析(Parsing)
- 逻辑查询计划生成(Logical Query Plan)
- 查询重写(Query Rewrite)
- 查询计划代价估计(Cost Estimation)
 - 中间结果大小估计 ← We are here !
 - I/O代价估计 ← Next
- 物理查询计划选择(Physical Query Plan)

2、I/O代价估计

■ 估计什么？

- 执行查询计划所必须读（写）的磁盘块数目

■ 需要另一些参数

- $B(R)$: R 所需的块数
- $f(R)$: 每块可容纳的 R 的最大元组数
- M : 可用的内存块数
- $HT(i)$: 索引 i 的层数
- $LB(i)$: 索引 i 的叶结点所需的块数

2、I/O代价估计

■ 影响查询计划I/O代价的因素

- 实现查询计划的逻辑操作符
 - ◆ 在选择逻辑查询计划时已确定
- 中间结果的大小 **already discussed!**
- 实现逻辑操作符的物理操作符 **Next**
 - ◆ 例如，连接操作是用索引连接还是散列连接？
- 相似操作的顺序 **see Sec.16.6**
 - ◆ 例如，多关系的连接顺序
- 物理操作符之间的参数传递方式 **see Sec.16.7**
 - ◆ **Pipeline**（流水线）还是**Materization**（物化）？

2、I/O代价估计

■ 物理操作符之间的参数传递

● 物化方式

- ◆ 操作依次执行，并且每个操作的结果（中间关系）都写到磁盘上供其它操作存取
- ◆ 通过磁盘物理进行数据传递
- ◆ 节省主存空间

● 流水线

- ◆ 多个操作同时执行，一个操作产生的元组直接通过共享内存传递给其它操作
- ◆ 节省I/O
- ◆ 但占用主存，若缓冲区出现“颠簸”则I/O增加

五、物理查询计划选择

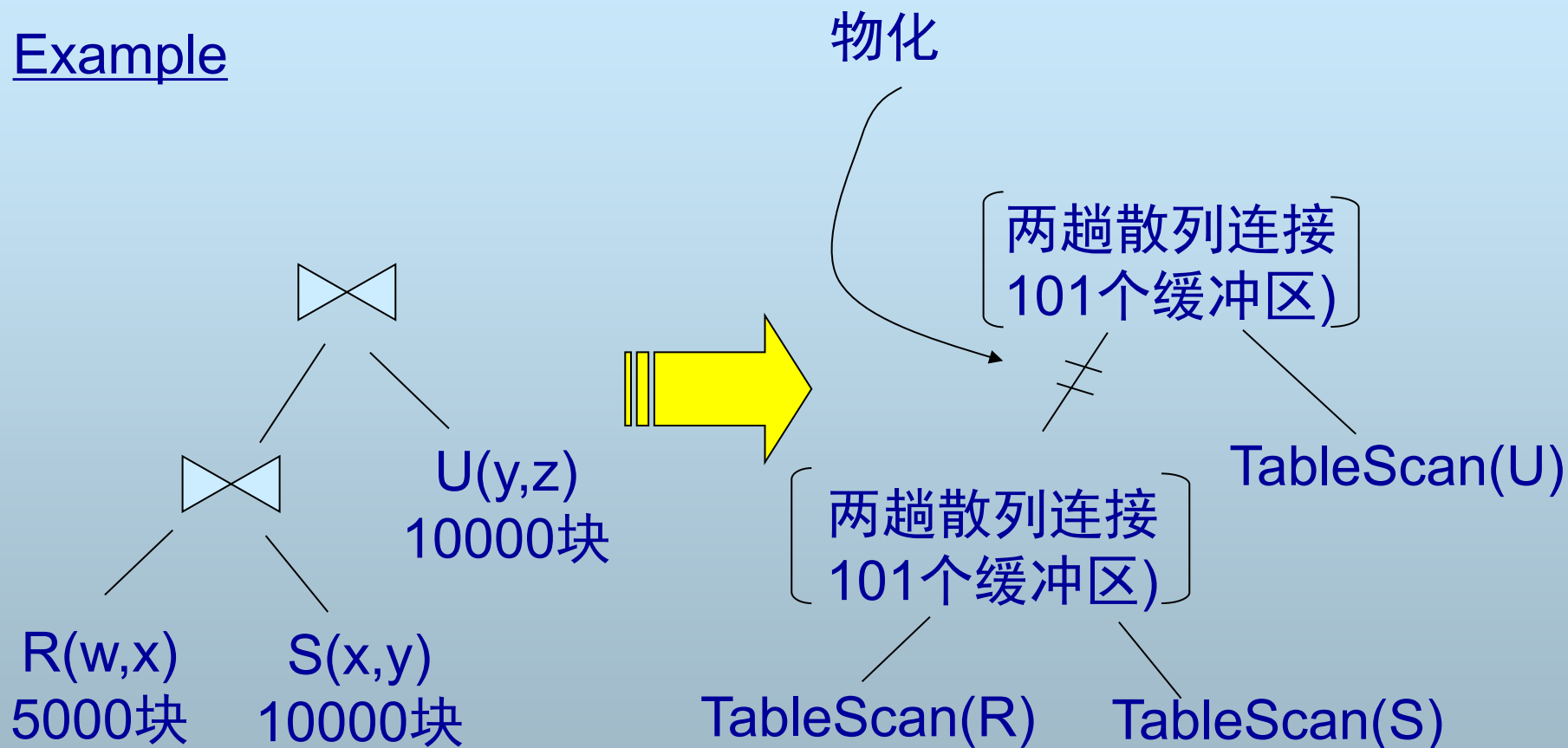
■ 物理查询计划生成

- 逻辑查询计划
- 估计中间关系大小
- 为逻辑操作符选择物理操作符
- 确定其它物理操作符
- 确定物理操作符的顺序
- 确定物理操作之间的参数传递方式

Next Chp.

五、物理查询计划选择

Example



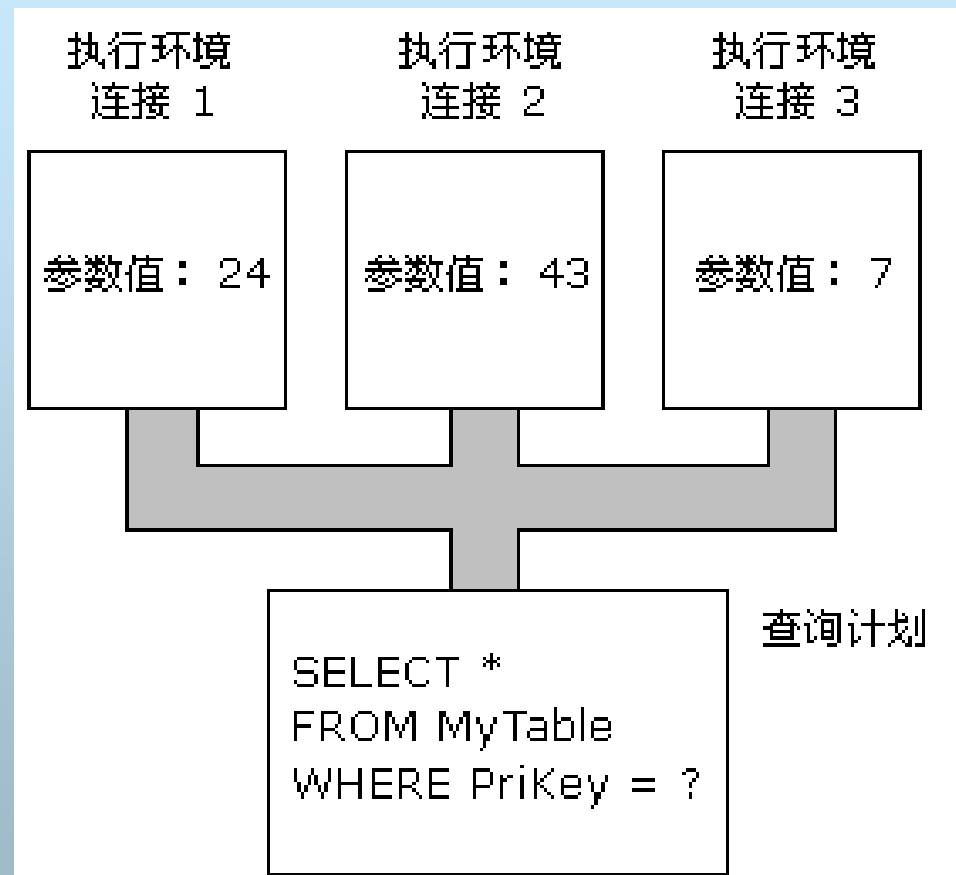
六、MS SQL Server的查询优化

- MS SQL Server查询优化过程
 - 语法分析
 - 查询树生成
 - 逻辑优化：逻辑查询计划等价转换
 - 基于代价的物理优化

1、执行计划

- **MS SQL Server 执行计划**包含下面两个主要组件：
 - **查询计划**：只读数据结构，可由任意数量的用户使用。这称为查询计划。
 - **执行环境**：每个正在执行查询的用户都有一个包含其执行专用数据（如参数值）的数据结构。该数据结构称为执行环境。执行环境数据结构可以重新使用。

1、执行计划



2、执行计划的缓存和重用

- 在 MS SQL Server 中执行任何 SQL 语句时，关系引擎将首先查看缓存中是否有用于同一 SQL 语句的现有执行计划。MS SQL Server 重新使用所找到的任何现有计划以节省重新编译 SQL 语句的开销。
- 如果没有现有执行计划，则 MS SQL Server 将为查询生成新的执行计划。

3、执行计划的老化

- 缓存中的每个执行计划都有相关的**编译成本因子**和一个**年龄字段**。执行计划每由连接引用一次，其年龄字段便按编译成本因子递增。
 - 例如，如果一个查询计划的成本因子是 **8** 且被引用了两次，它的年龄将变为 **16**。
- **惰性写入器进程**定期扫描缓存内的对象列表。每扫描一次将年龄字段减少 **1**。如果满足下面三个条件，惰性写入器进程将释放对象：
 - 内存管理器需要内存且所有可用内存都正在使用。
 - 对象的年龄字段是 **0**。
 - 对象在当前没有被连接引用。

4、执行计划的重编译

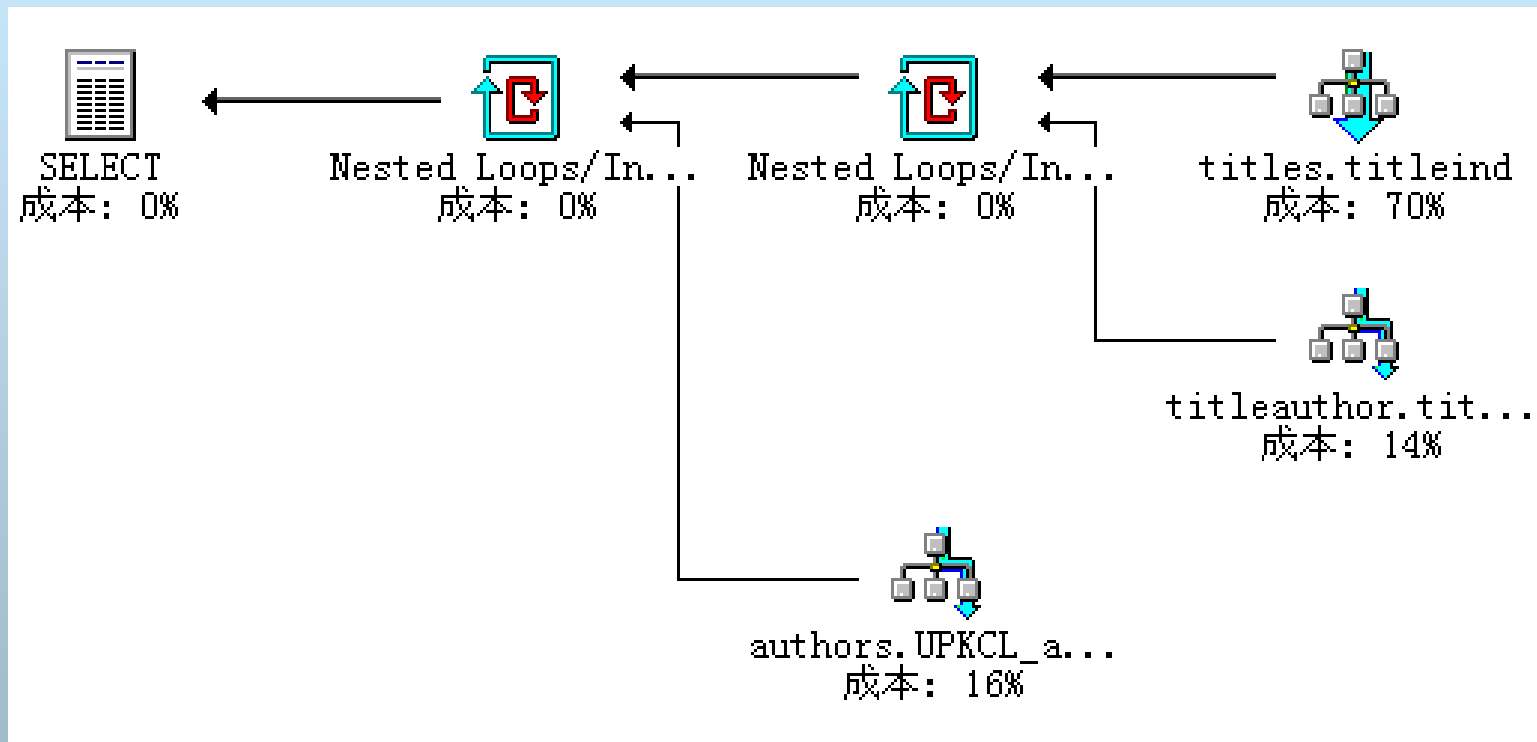
■ 需要重编译的情况

- 对查询所引用的表进行任何结构更改。
- 通过语句（如 **UPDATE STATISTICS**）显式生成或者自动生成新的统计信息。
- 删除执行计划所使用的索引。
- 显式调用 **sp_recompile**。
- 对键的大量更改（其他用户对由查询引用的表使用 **INSERT** 或 **DELETE** 语句所产生的修改）。
- 对于带触发器的表，**inserted** 或 **deleted** 表内的行数显著增长。

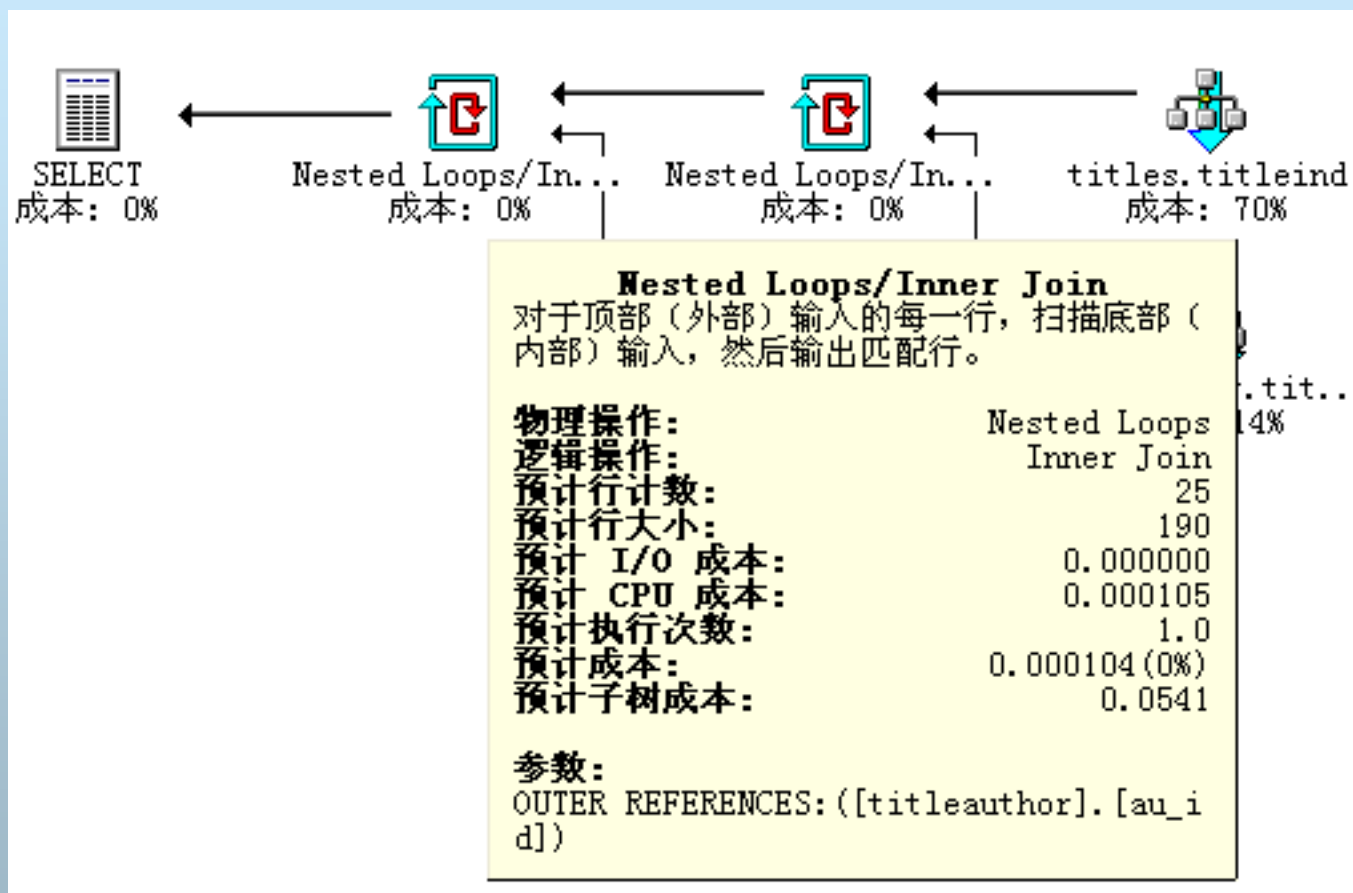
5、执行计划例子

```
SELECT authors.au_fname, titles.title  
FROM authors, titles, titleauthor  
WHERE authors.au_id = titleauthor.au_id AND  
       titleauthor.title_id = titles.title_id
```

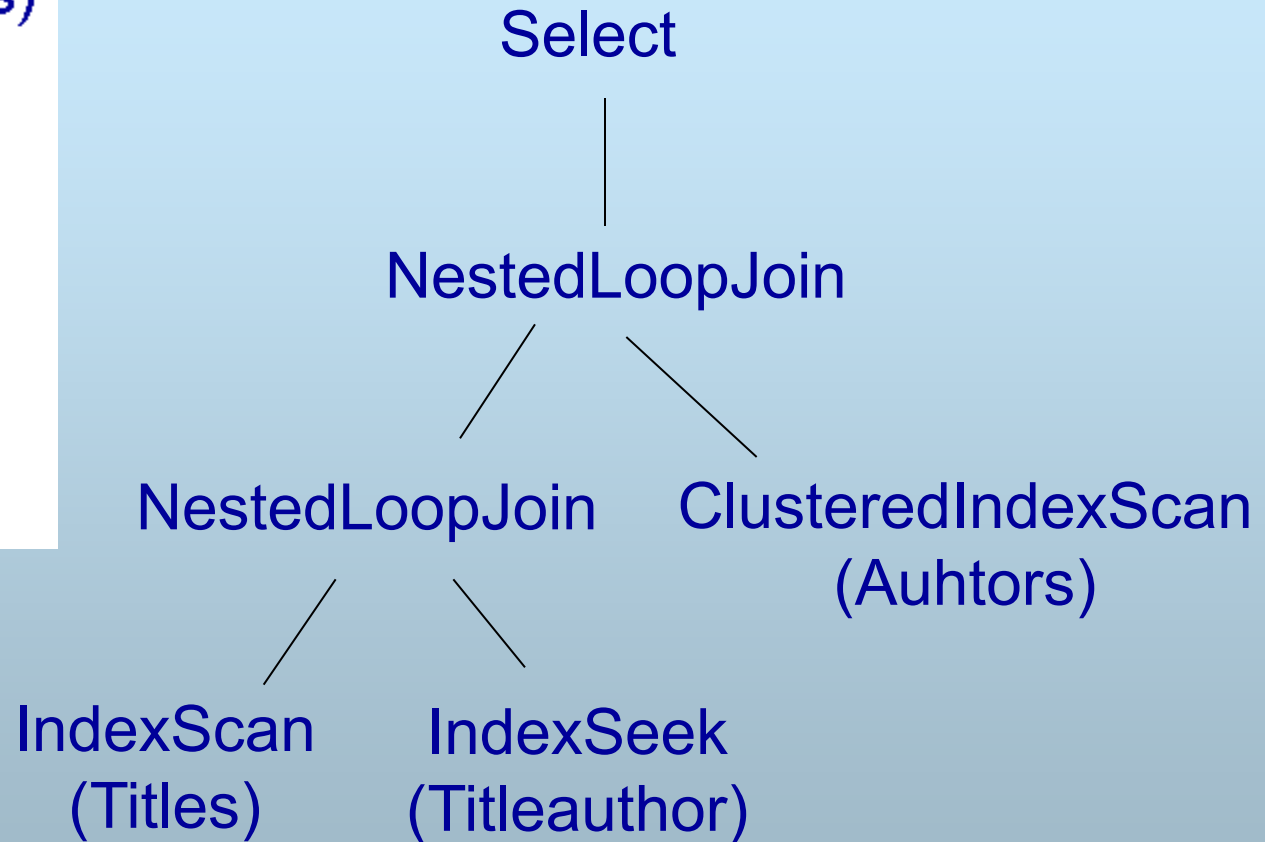
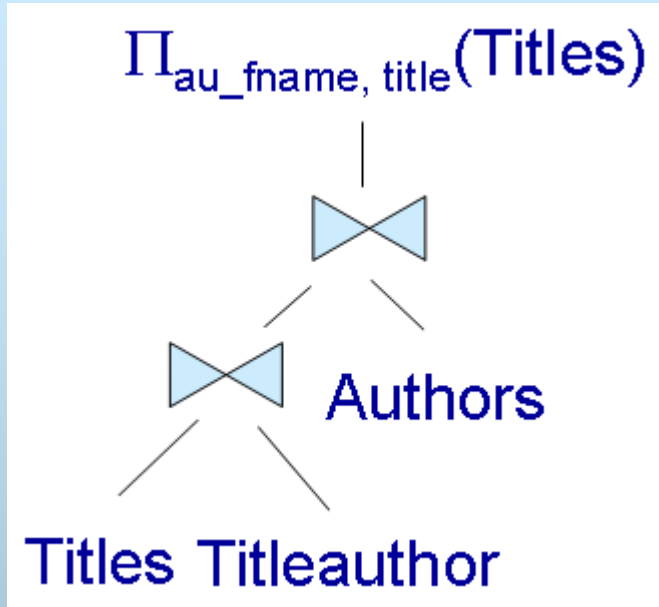
5、执行计划例子



5、执行计划例子



5、执行计划例子

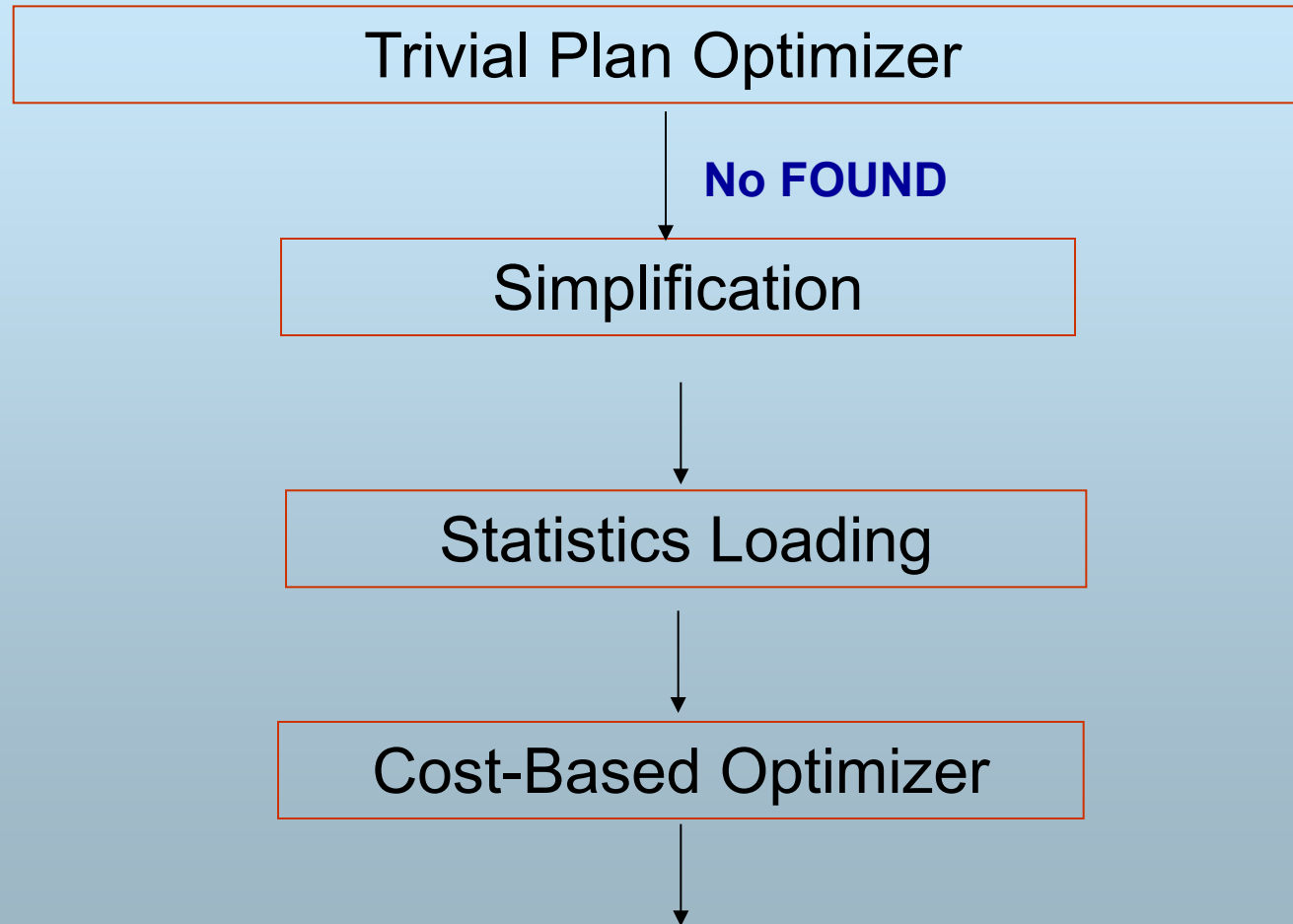


6、MS SQL Server查询优化器

■ 多阶段优化器

- **Trivial Plan阶段**：某些查询只有一个执行计划（如Insert），优化器只需产生一个计划
- **Simplification阶段**：语法转换
- **Statistics Loading阶段**：装载代价估计所需的统计信息
- **Cost-Based Optimization阶段**：分阶段运用转换规则生成执行计划，并估计代价
 - ◆ 事务处理查询阶段：使用简单转换规则
 - ◆ 复杂查询阶段：使用包括连接、分组等的转换规则

6、MS SQL Server查询优化器



小结

- **语法分析(Parsing)**
- **逻辑查询计划生成(Logical Query Plan)**
- **查询重写(Query Rewrite)**
- **查询计划代价估计(Cost Estimation)**
- **物理查询计划选择(Physical Query Plan)**