



大数据计算基础

第五章 大数据管理系统

王宏志

wangzh@hit.edu.cn

<http://homepage.hit.edu.cn/pages/wang>

- 1 大数据管理概述
- 2 数据库管理系统
- 3 NoSQL
- 4 NewSQL

1 大数据管理概述

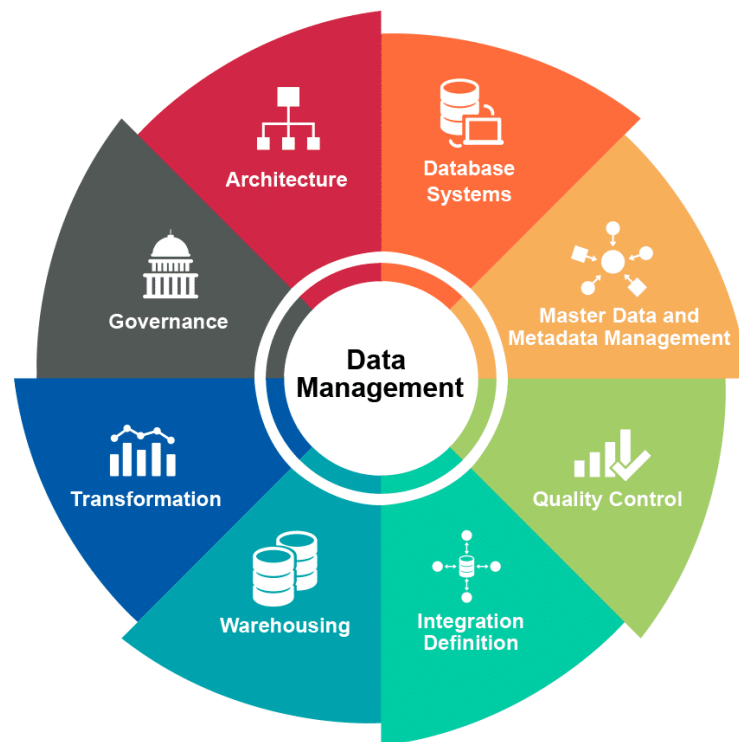
2 数据库管理系统

3 NoSQL

4 NewSQL

为什么要管理数据？

- 数据管理是利用计算机硬件和软件技术对数据进行有效的收集、存储、处理和应用的过程。其目的在于充分有效地发挥数据的作用。
- 实现数据有效管理的关键是数据组织。



• 人工管理阶段

- 20世纪50年代中期以前，计算机主要用于科学计算
- 外存只有磁带、卡片、纸带，没有磁盘等直接存取的存储设备
- 没有操作系统，没有管理数据的软件
- 数据处理方式是批处理
- 数据不保存、数据无专门软件进行管理、数据不共享、数据不具有独立性、数据无结构

• 文件系统阶段

- 20世纪50年代后期到60年代中期，计算机开始用于管理
- 硬件方面已经有了磁盘、磁鼓等直接存取的存储设备
- 软件方面，操作系统中已经有了数据管理软件，一般称为文件系统
- 处理方式上不仅有了文件批处理，而且能够联机实时处理

• 数据库系统阶段

- 20世纪60年代末数据管理进入新时代——数据库系统阶段
- 数据库系统阶段出现了统一管理数据的专门软件系统，即数据库管理系统
- 数据库系统是一种较完善的高级数据管理方式，也是当今数据管理的主要方式，获得了广泛的应用。

◆ 分布式数据库

◆ 演绎数据库

◆ 并行数据库

◆ 面向对象数据库

◆ 对象-关系数据库

◆ Web数据库

◆ XML数据库

大数据时代，数据管理做什么？

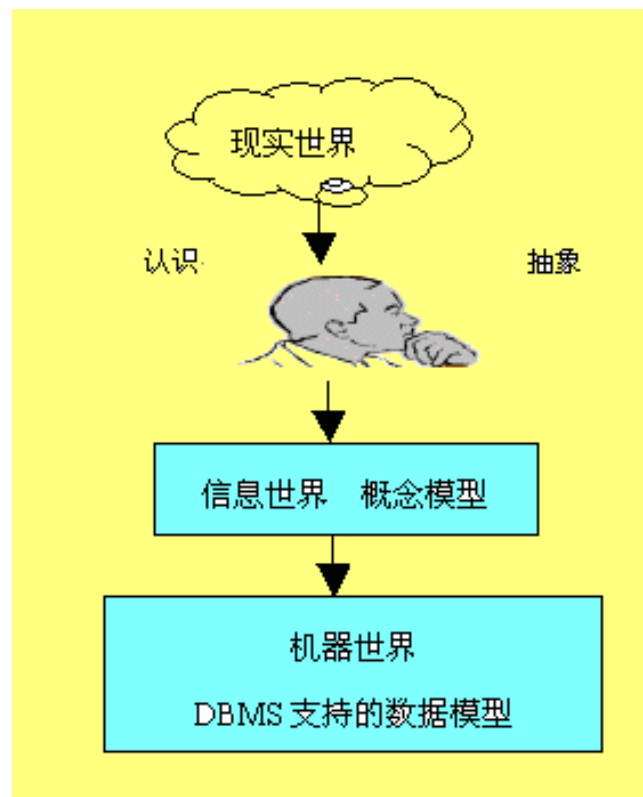
- 数据治理
- 数据架构
- 数据模型与设计
- **数据库与存储**
- 数据安全和隐私保护
- 主数据与元数据管理
- 数据集成和互操作
- 文档与内容管理
- 数据仓库与数据挖掘
- 数据质量

1 大数据管理概述

2 数据库管理系统

3 NoSQL

4 NewSQL



三个世界的转化过程示意图

- 人们收集并抽取出一个应用所需要的大量数据之后，应将其保存起来以供进一步加工处理，进一步抽取有用信息
- 数据库的定义
 - 数据库(Database,简称DB)是长期储存在计算机内、有组织的、可共享的数据集合。

- 数据库的特征
 - 数据按一定的数据模型组织、描述和储存
 - 可为各种用户共享
 - 冗余度较小
 - 数据独立性较高
 - 易扩展

- 什么是DBMS

- 数据库管理系统 (Database Management System , 简称DBMS) 是位于用户与操作系统之间的一层数据管理软件。

- DBMS的用途

- 科学地组织和存储数据、高效地获取和维护数据

- DBMS的主要功能

- 数据定义功能:提供数据定义语言(DDL), 用于定义数据库中的数据对象。
- 数据操纵功能:提供数据操纵语言(DML), 用于操纵数据实现对数据库的基本操作(查询、插入、删除和修改)。

- DBMS的主要功能(续)

- 数据库的运行管理:保证数据的安全性、完整性、多用户对数据的并发使用及发生故障后的系统恢复。
- 数据库的建立和维护功能:提供实用程序,完成数据库数据批量装载,数据库转储,介质故障恢复,数据库的重组织和性能监视等

- 什么是数据库系统
 - 数据库系统（ Database System，简称DBS）是指在计算机系统中引入数据库后的系统构成。
 - 在不引起混淆的情况下常常把数据库系统简称为数据库。
- 数据库系统的构成
 - 由数据库、数据库管理系统（及其开发工具）、应用系统、数据库管理员和用户构成。

- 存储介质

- 内外存数据交换

- 数据库引擎中的存储管理器，主要两个部件：缓冲区管理器和文件管理器。
 - 数据库运行时，内外存间要频繁地进行数据交换，每交换一次数据称为一次I/O操作。数据库系统需OS支持进行I/O。
 - I/O操作是很慢的，数据调出外存要缓存在缓冲区中供后续的操作使用(读/写)。读不命中或写结果按策略更新时进行I/O。
 - 每次交换的数据量称为一个数据块，一个块可以等于一个或几个磁盘块。块大对顺序访问有利，对随机访问不利。
 - 内存中缓冲区的大是若干个数据块。

- 磁盘冗余阵列(RAID)

- RAID是数据库服务器最常用的外存储介质，有若干同样的磁盘组成的阵列，从RAID0-RAID8有多种组合方式。
- 通过冗余改善可靠性。
 - » 同一数据同时写入两个磁盘
 - » 若干个磁盘数据用一个磁盘保存校验位。
- 通过数据条块化提高速度
 - » 数据大体均匀地存放于若干磁盘上。
 - » 当多用户请求数据库数据时，I/O操作可以随机地落在不同的磁盘，磁盘阵列并行I/O从而提高速度。

- 备份数据和历史数据

- 通过网络存储存放到物理上分离的地方(防火、水、震等)
- 移动硬盘或光盘、磁带等大容量离线存储设备。

– 文件内记录的存储

- 数据库数据以文件形式在外存中存储。数据库的逻辑记录在物理文件中如何实现，是记录的存储方法问题。
- 定长记录格式：每个数据库记录占有定长文件记录。

– 例：Employer(ENAME char(10),ENO char(10),SALARY real)

设一个实数占8字节，
则一个记录28字节，
文件的逻辑机构为：

– 删除操作空位处理

- » 其它记录依次上移
- » 最后记录填补空位
- » 被删空位链接、插入时用

记录0	Liu	A-102	6000
记录1	Wen	B-306	7000
记录2	He	F-257	8000
记录3	Zhang	A-214	6000
记录4	Zhou	C-343	7500
记录5	Liu	B-215	8000

- 变长纪录：每条数据库记录长度不同
 - 例：Salsepson的属性adress是变长的
Salseperson(empID:char(3),empName:char(8),sex:char(1),birthday:date,spTelNo:number(12),address:varchar(50))
 - 字符串格式：把每个数据库记录看作是连续的字节串，尾部加记录尾标记符。(大多数操作系统支持按行存取文件)
 - 下面的关系

empID	empName	sex	Birthday	spTelNo	address
sp1	刘女士	F	70-1-20	082354	北京市宣武区牛街街道双槐里小区22楼2-101 100028
sp2	李先生	M	67-5-2	213146	北京市海淀区玉泉路甲19号 100049
sp3	何女士	F	62-9-7	352210	山东省日照市 276534

- 下述文件存储是字符串格式

```
0 sp1 | 刘女士 | F | 70-1-20 | 1390082354 2 | 北京市宣武区牛街  
    街道双槐里小区22楼2-101 | 1370100028 1 | ⊥ |  
1 sp2 | 李先生 | M | 67-5-22 | 13146092228 | 北京市海淀区玉  
    泉路甲19号100049 | ⊥ |  
2 sp3 | 何女士 | F | 62-9-7 | 13352210756 | 山东省日照市 2 76  
    534 | ⊥ |
```

- 字符串格式的缺点

- 被删除的位置难以重新利用。
- 某记录要伸长很困难(移动大量记录)。

- 改进：分槽式页结构(shotted page structure)

- 记录从块的尾部邻接存放。中间是自由空间(伸长记录用)。
- 块的开始是块首部，记录块中记录数、每个记录大小和位置。

数据库存储结构

• 分槽式变长记录结构



• 变长记录的定长表示

- 预留空间(按最大)
- 固定块+溢出块格式

固定块

sp1	刘女士	F	70-1-20	082354	
sp2	李先生	M	67-5-2	213146	
sp3	何女士	F	62-9-7	352210	

溢出块

北京市宣武区牛街街道	
双槐里小区22楼2-101	
100028	⊥
北京市海淀区玉泉路甲	
19号100049	⊥
山东省日照市 276534	⊥

– 文件内记录的组织

- 一个文件包含了成千上万个记录，这些记录按什么顺序或方式安排，是数据库记录在文件内的组织问题。
- 对某种组织的文件怎样去查找、插入和删除，是文件中记录的存取方法问题。不同的组织方式存取效率有很大差别。
- 记录的组织方式
 - 堆文件组织：记录可以放在文件的任何位置，以输入顺序为序。删除、插入操作不需移动数据。
 - 顺序文件组织：记录按查找键值的升序或降序的顺序逻辑存储的。一般使用指针链结构。
 - 散列文件组织(hashing file)：某个属性值通过哈希函数求得的值作为记录的存储地址。
 - 聚类文件组织：一个文件可存储多个有联系的关系，有联系的记录存储在同一块内，以提高I/O速度。

- 顺序文件组织

- 记录的物理顺序和查找键值一致：插入、删除需移动数据。

记录0	He	F-257	800
记录1	Liu	B-215	8000
记录2	Liu	A-102	6000
记录3	Wen	B-306	7000
记录4	Zhang	A-214	6000
记录5	Zhou	C-343	7500

记录0	He	F-257	800	
记录1	Liu	B-215	8000	
记录2	Liu	A-102	6000	
记录4	Wen	B-306	7000	
记录5	Zhang	A-214	6000	
记录6	Zhou	C-343	7500	

Ma	B-547	500	
----	-------	-----	--

- 用指针逻辑链接：

- » 插入：找到键值位置；记录3
 - 插入到空闲位置；修改指针；
 - » 删除：修改指针；回收空闲位置。

- 聚类文件组织

- 聚类文件的组织方式与查询的类型有关，一个文件内有两个或多个关系的记录类型。

- 例：教学数据库中的关系S和SC，经常进行自然连接查询操作，如 `SELECT S.S#,Sname,C#,Grade`

`FROM S,SC WHERE S.S#=SC.S#` 聚类文件

- 当数据量很大时，两个文件内记录的连接操作是很慢的。聚类文件格式如右图：

- 关系S和SC

S#	Sname	Age	Sex
S1	Liu	21	F
S2	Wang	20	M
S3	Chen	22	M

S#	C#	Grade
S1	C1	80
S1	C2	70
S3	C1	90
S3	C2	85
S3	C3	95

记录1
记录2

S1	Liu	21	F
S1	C1	80	
S1	C2	70	
S2	Wang	20	M
S3	Chen	22	M
S3	C1	90	
S3	C2	85	
S3	C3	95	

– 索引技术

- 当文件中记录的数目和数据量很大时，直接查找速度会很慢。必须建立索引机制。
- 索引是独立于主文件记录的一个只含索引属性的小的文件，且按索引值排序，查找速度可很快。常用的索引或一、二级索引可以读入缓冲区以加快速度。
- 索引分类
 - 有序索引：根据记录中某种排序顺序建立的索引。
 - 散列索引：根据记录中某个属性值，通过散列函数得到值作为存储空间的桶号。
- 有序索引分类：主文件可建立几个索引文件。
 - 主索引(聚类索引)：索引的查找键值的顺序与主文件顺序一致。这种文件称作索引顺序文件。主索引只有一个。
 - 非聚类索引：索引的查找键值的顺序与主文件顺序不一致。

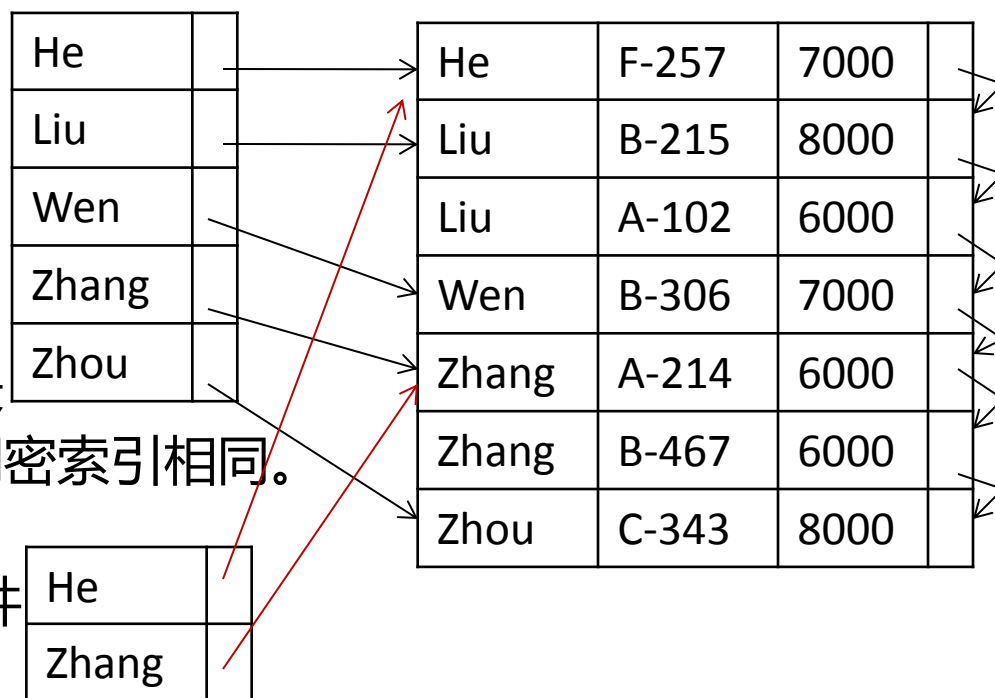
- 主索引

- 稠密索引：对于主文件中的每一个查找键值建立一个索引记录(索引项)，索引记录包括查找键值和指向具有该值的主文件中第一个记录的指针。

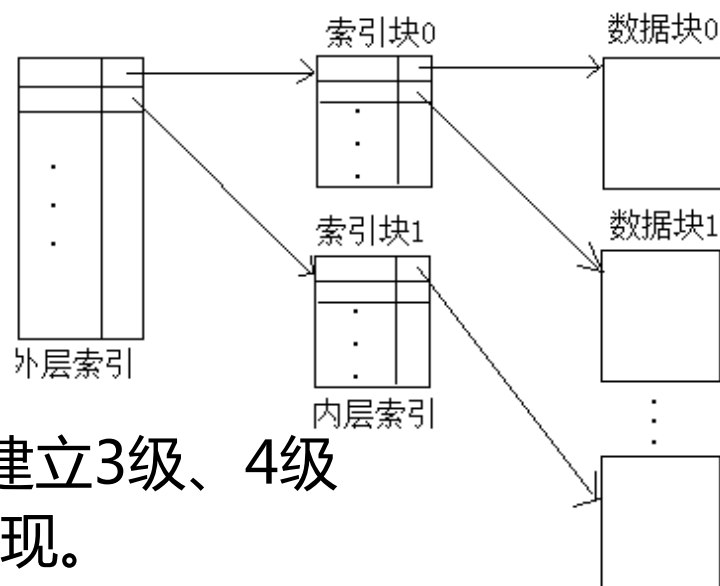
- 例：

- 稀疏索引：在主文件中，若干个查找键值才建立一个所以记录索引记录的内容与稠密索引相同。

- 例：要找到Liu，先找到He指向的主文件然后顺链找到Liu。



- 多级索引：即使采用稀疏索引，可能建成的索引还是很大，以致于查询效率不高。
 - 例：主文件100 000个记录，每块可存储10个记录，需10 000个数据块。若以块为单位建立稀疏索引，索引需10 000项。虽然索引记录比主记录小得多，如每块可存100个索引项，索引块仍需100块，缓冲区就很难放得下。
 - 解决的办法是对主索引再建立一级索引，如图所示的二级索引。
 - 如果外层索引数据量还是太大，可建立3级、4级索引。多级索引可用B+树或B树实现。



- 索引的更新

- 删除：为了在主文件中删除一条记录，需

- ①找到主文件记录，删除。

- ②如果符合索引键值的记录有多个，索引不用修改。

- ③否则，对稠密索引，删除相应的索引项；对稀疏索引，如果被删记录的索引值在索引块中出现，则用主文件被删记录的下一个记录的查找键A替换，若A已出现在索引块，则删除被删记录的对应多音键。

- 插入：

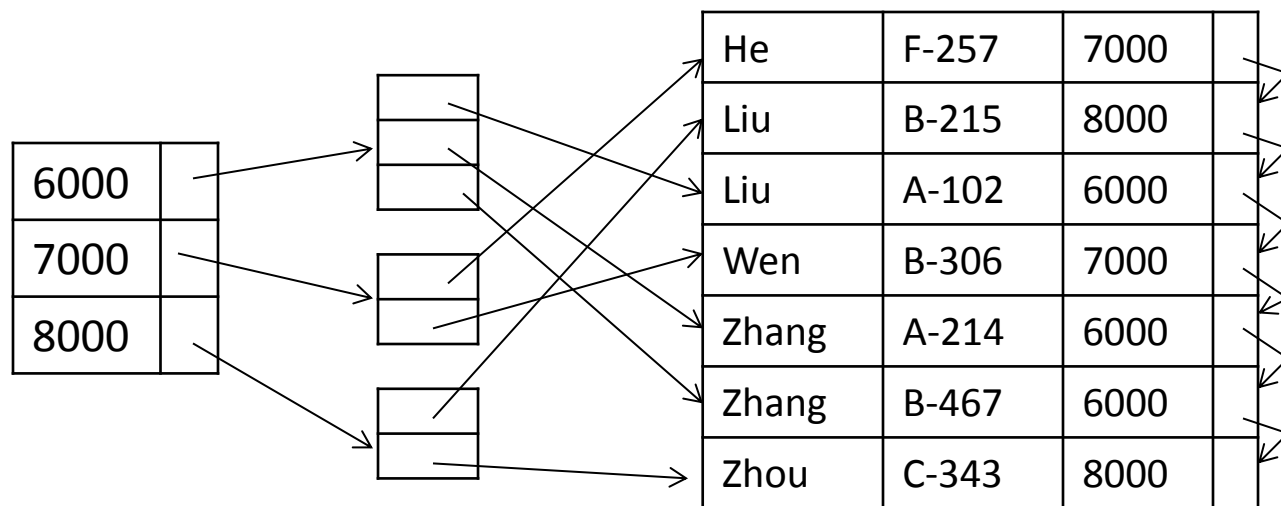
- ①用插入记录的查找键找到插入位置，执行主文件插入。

- ②对稠密索引且查找键未在索引块出现，在索引中插入。

- ③对稀疏索引，每块数据对应一个索引项。若数据块有空闲放得下新数据，不用修改索引；否则，加入新数据块，在索引块中插入一个新索引项。

• 辅助索引

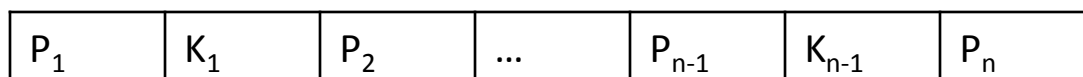
- 在其他属性上建立索引，也可以提高以这些属性为查找键值的查找速度，称为辅助索引。但由于相同查找键值的记录分散在主文件中(已按主索引顺序存储)，辅助索引需新的结构。
- 辅助索引的指针不直接指向主文件记录，而是指向一个桶，桶内存放指向具有同一查找键值的主记录指针。
- 辅助索引(最末级)不能是稀疏索引。
- 例：在Salary建索引



• B+树索引

– B+树的结构：一颗M阶的B+树，按下列方式组织：

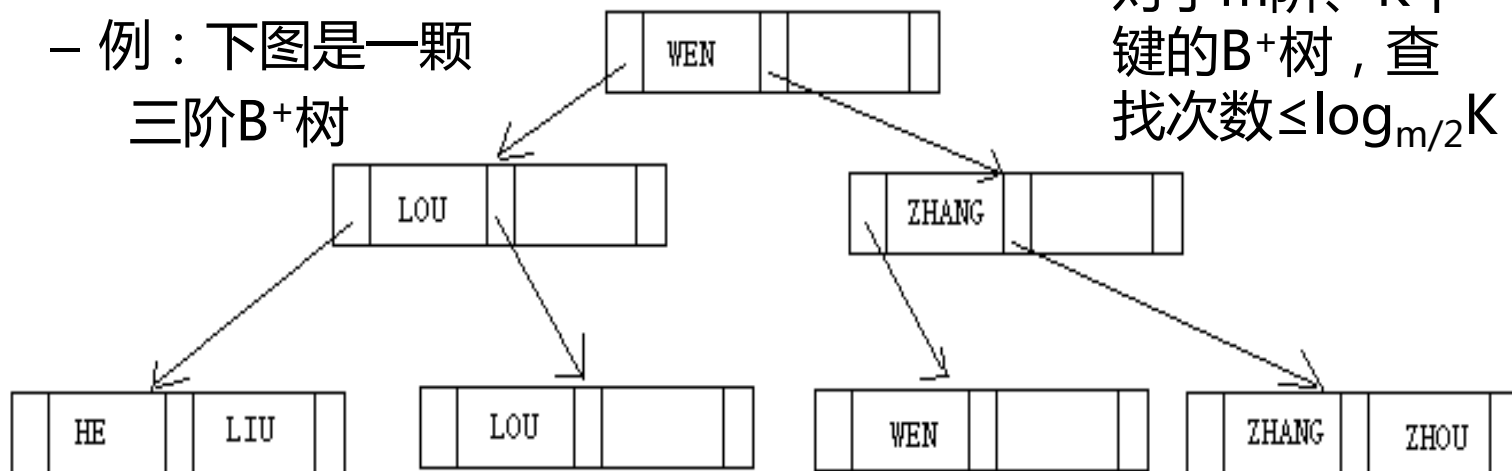
①每个结点至多有 $m-1$ 个查找键值和 m 个指针。如下图



②叶结点指针指向主文件中的记录或桶。

③非叶结点指针数 $m/2 \leq n \leq m$ ， P_i 指向的子树中所有键值均小于 K_i ，而大于等于 K_{i-1} 。

– 例：下图是一颗
三阶B+树



查询 → 查询处理

举例

Select B,D

From R,S

Where $R.A = \text{"c"} \wedge S.E = 2 \wedge R.C = S.C$

举例

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

举例

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

查询结果

B	D
2	x



一个办法

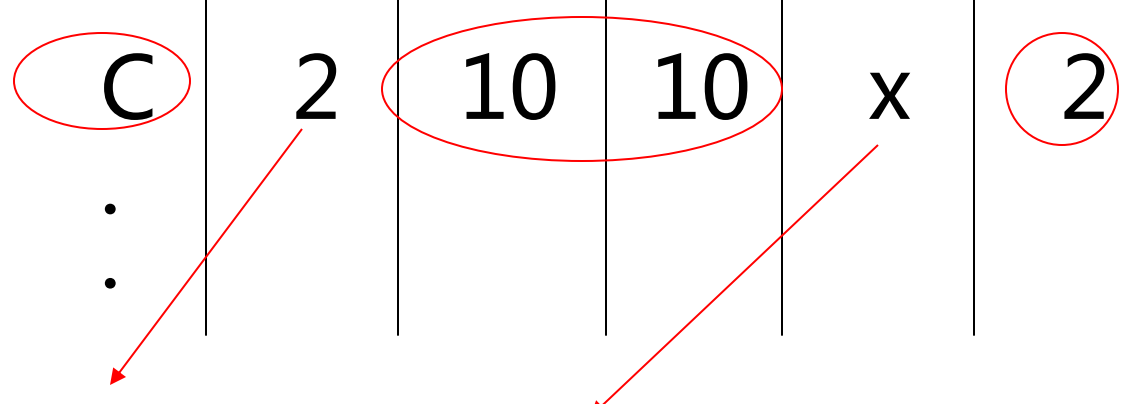
- 使用笛卡尔积
- 选择元组
- 使用映射

如何进行查询？

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
	.					
	.					

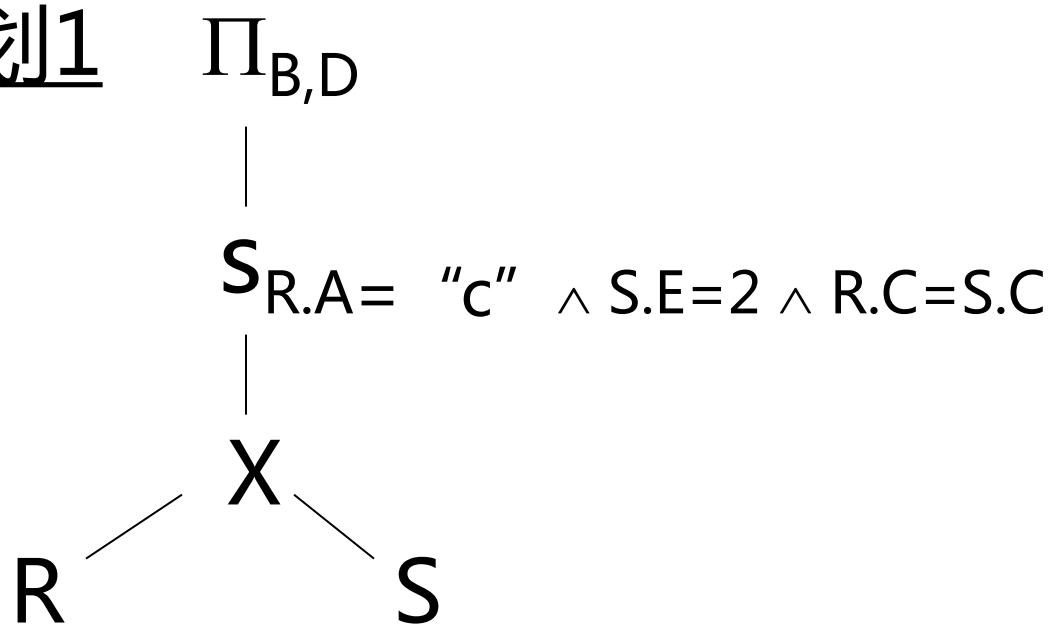
如何进行查询？

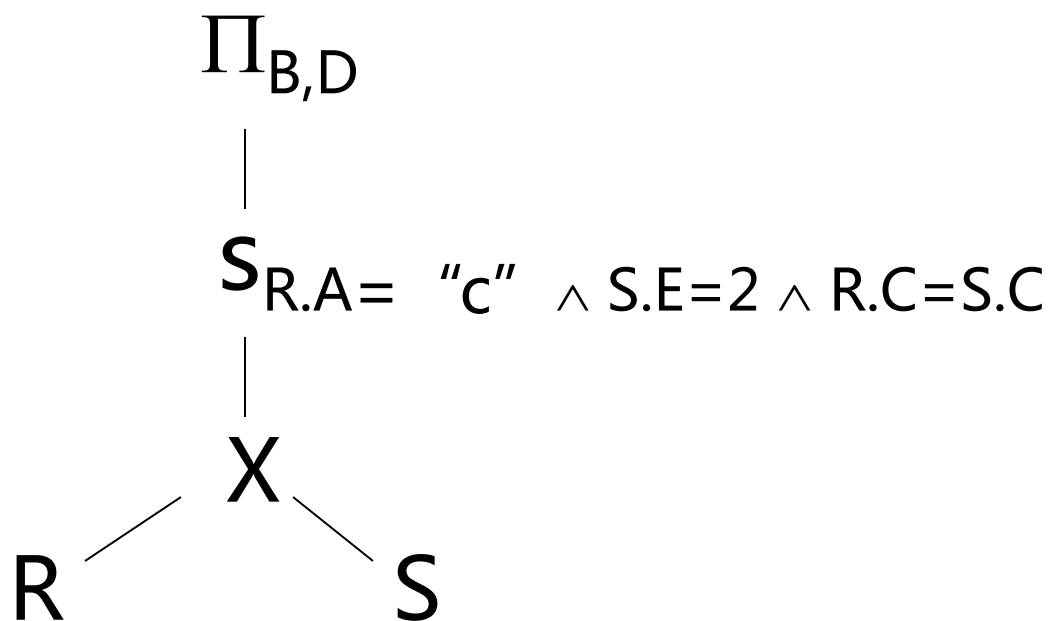
RXS	R.A	R.B	R.C	S.C	S.D	S.E
	a	1	10	10	x	2
	a	1	10	20	y	2
	.					
	.					
搞定! → 找到一个...	C	2	10	10	x	2
	.					
	.					



可被用于描述关系.....

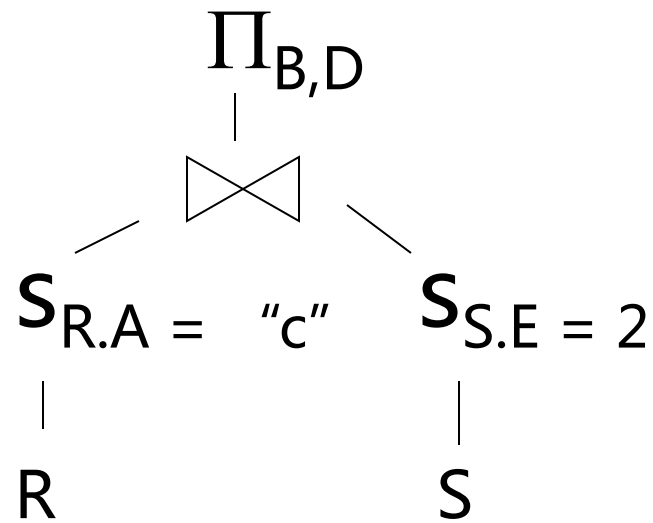
举例：计划1

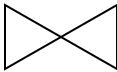


举例：计划1

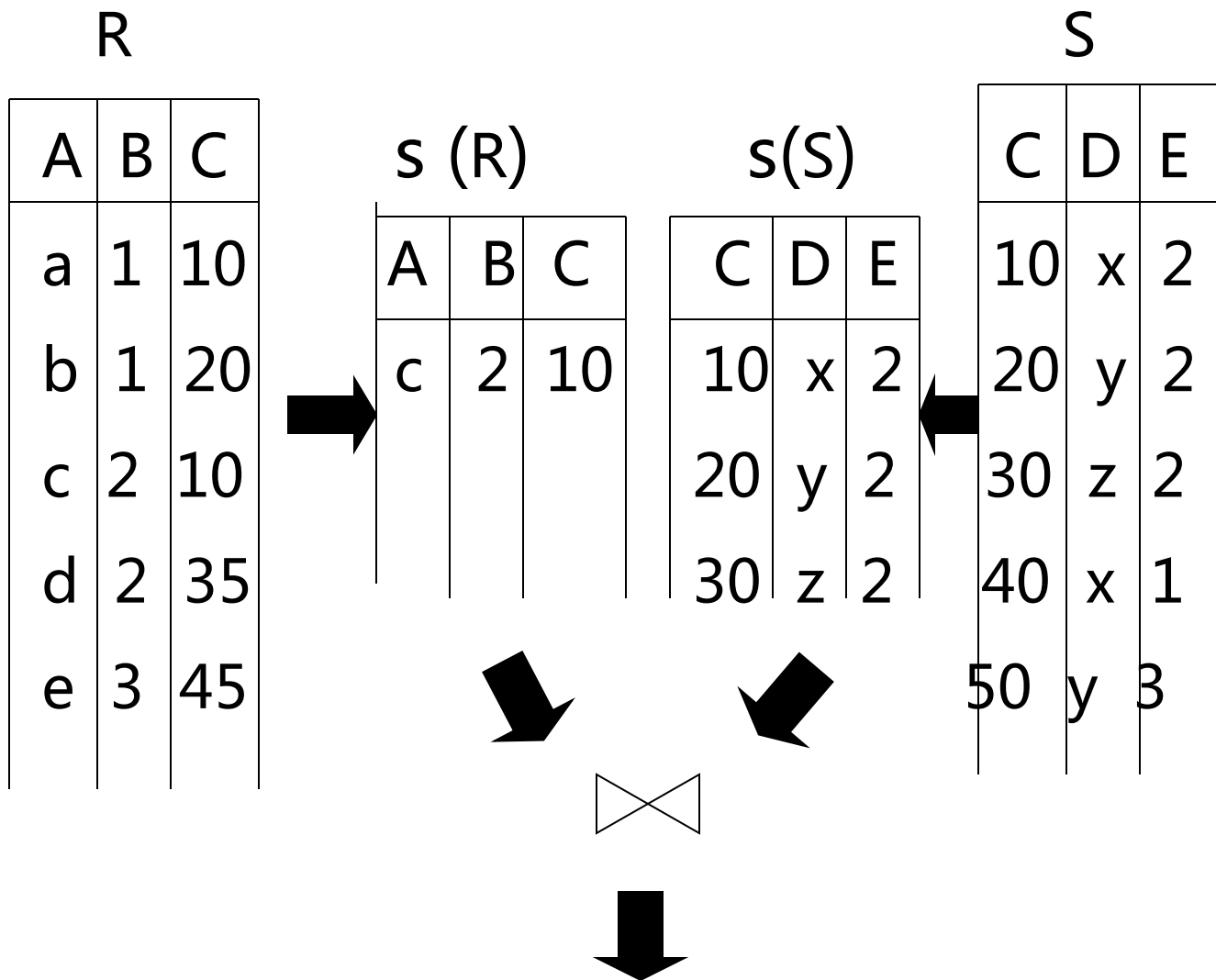
或: $\Pi_{B,D} [S_{R.A = "c" \wedge S.E = 2 \wedge R.C = S.C} (RXS)]$

计划2




自然连接

另一个想法



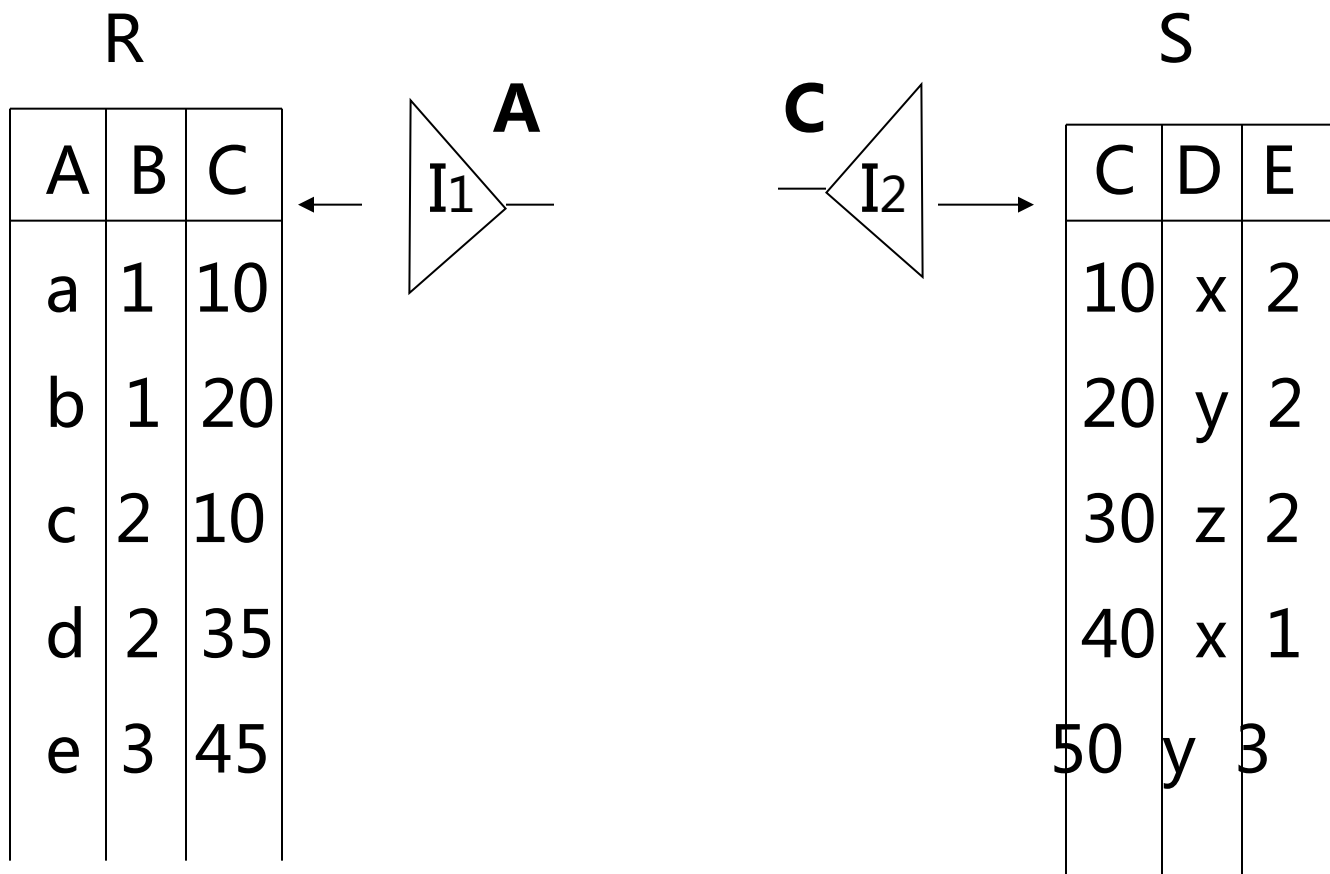
使用R.A和S.C索引

- (1) 使用R.A索引，令R.A= "c" 来选择R元组
- (2) 对于每一个找到的R.C值，使用S.C索引来找到匹配的元组

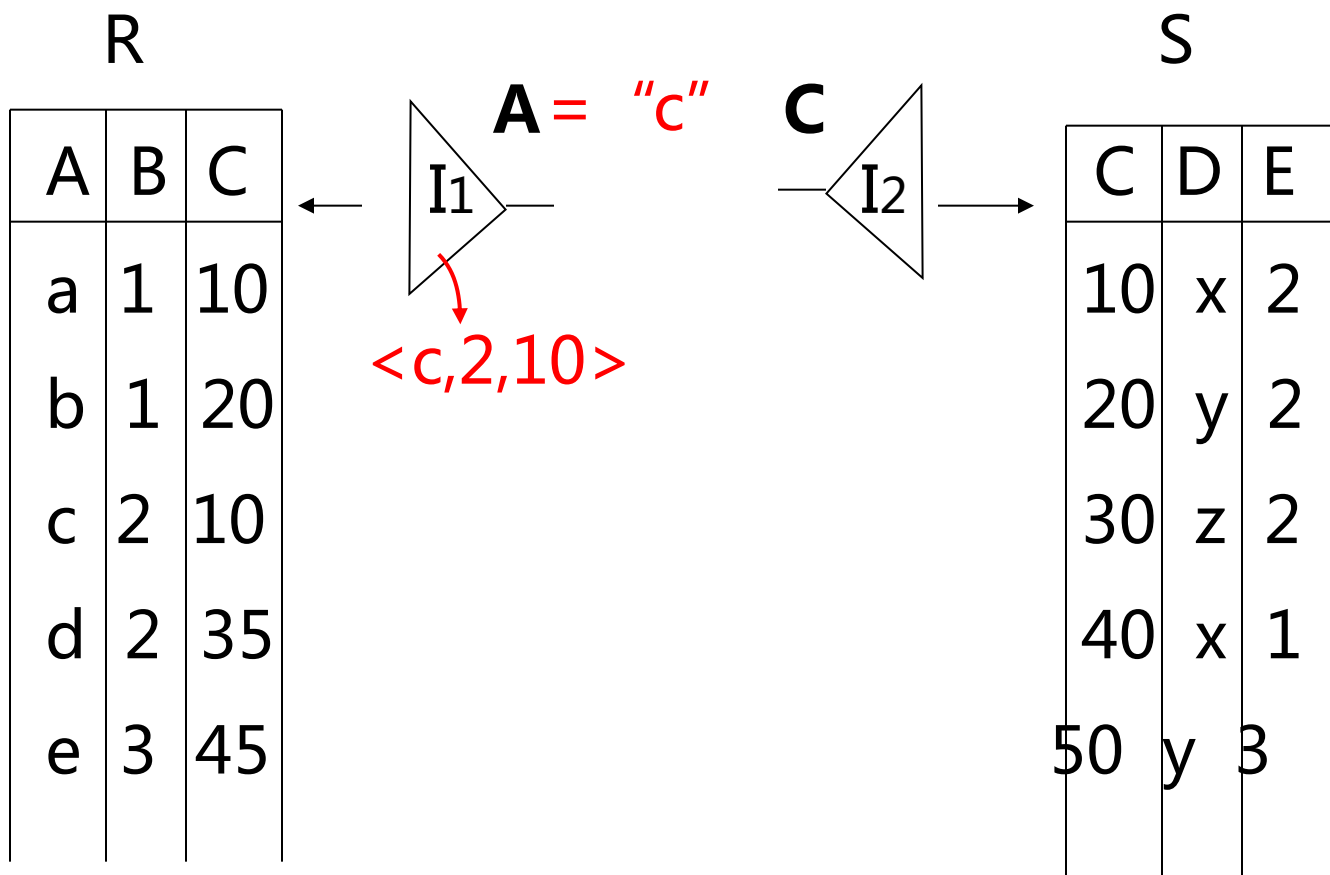
使用R.A和S.C索引

- (1) 使用R.A索引，令 $R.A = "c"$ 来选择R元组
- (2) 对于每一个找到的R.C值，使用S.C索引来找到匹配的元组
- (3) 排除 $S.E \neq 2$ 的S元组
- (4) 连接匹配的R、S元组和项目B、D的属性并放入结果中

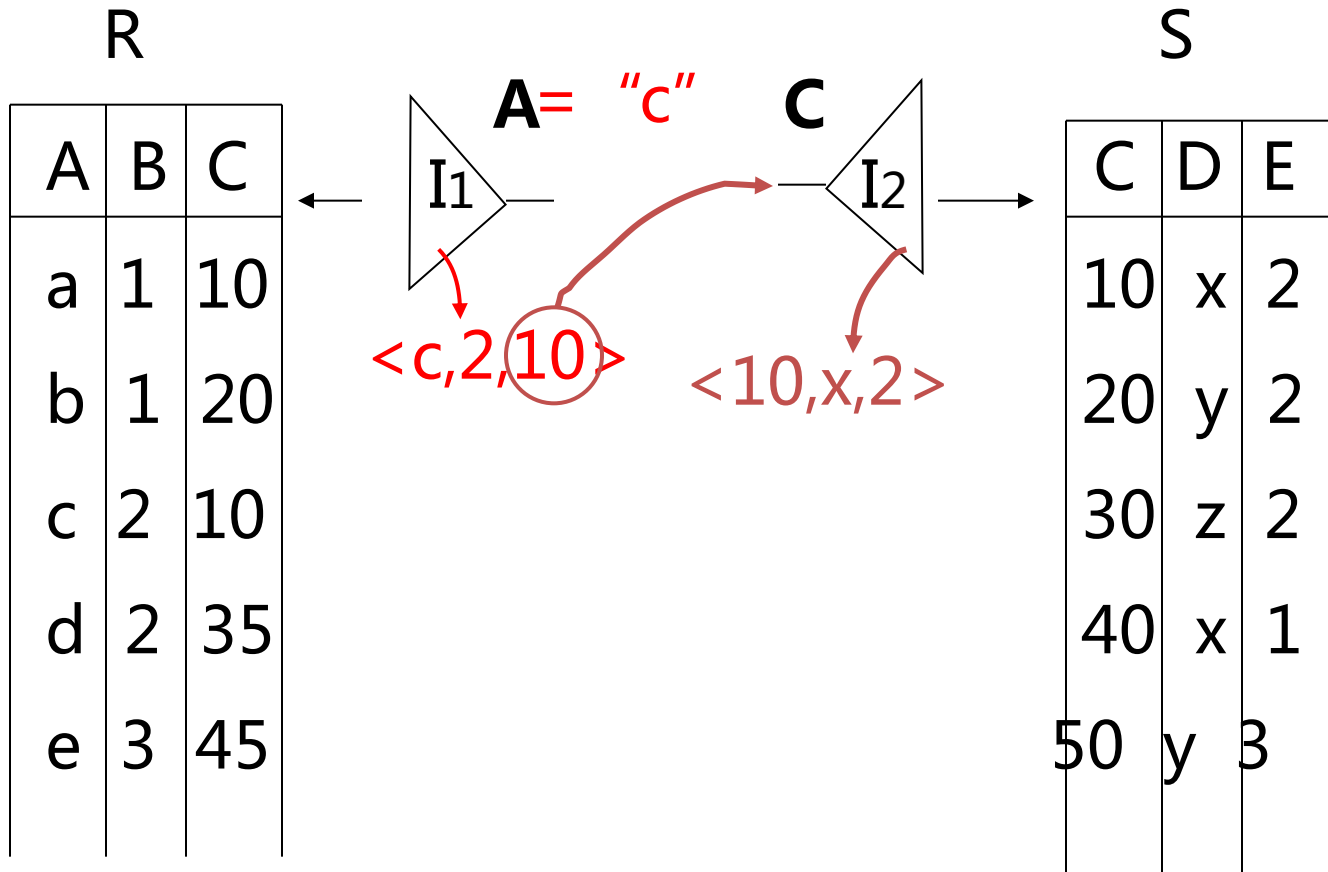
计划3



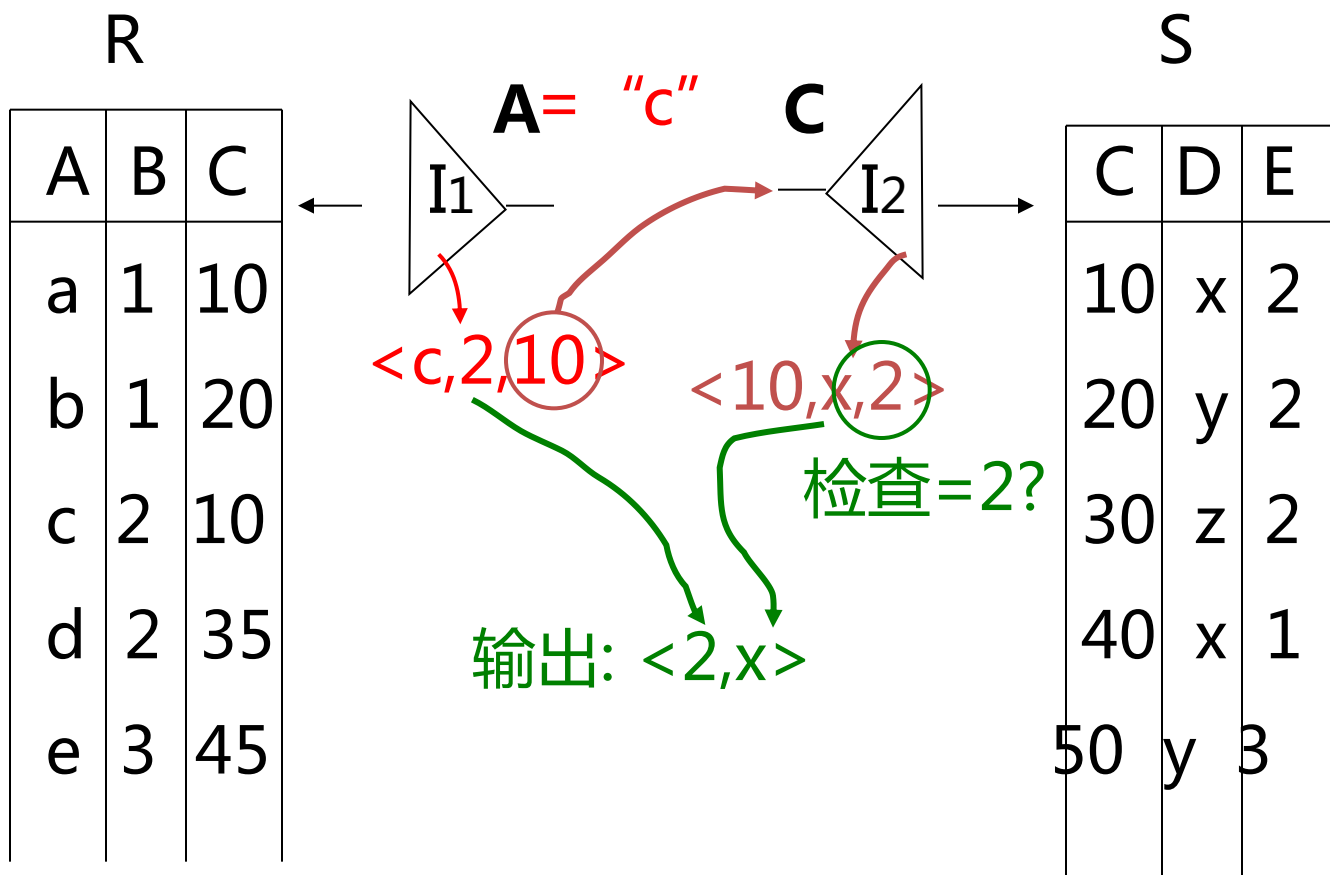
计划3



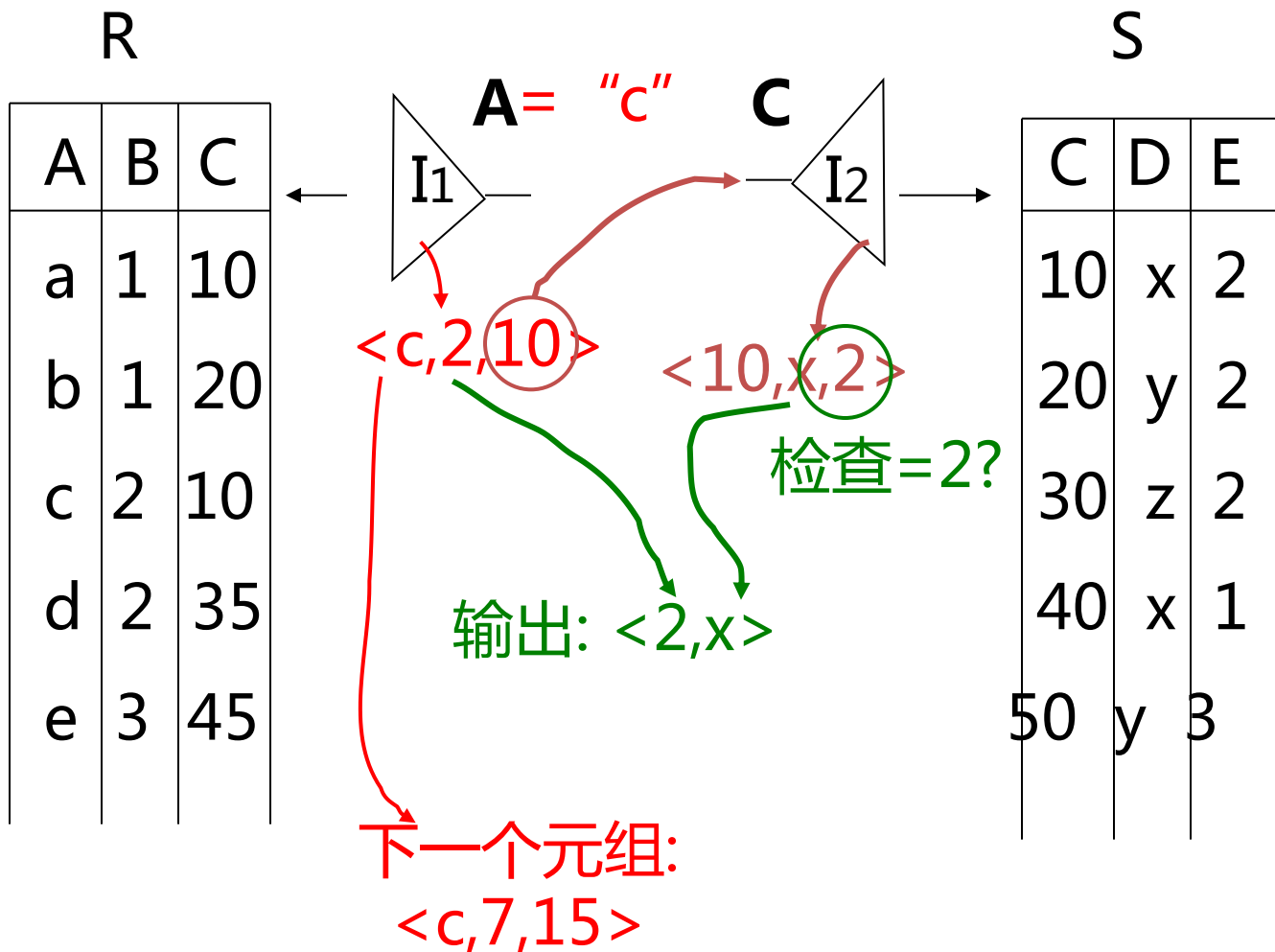
计划3



计划3

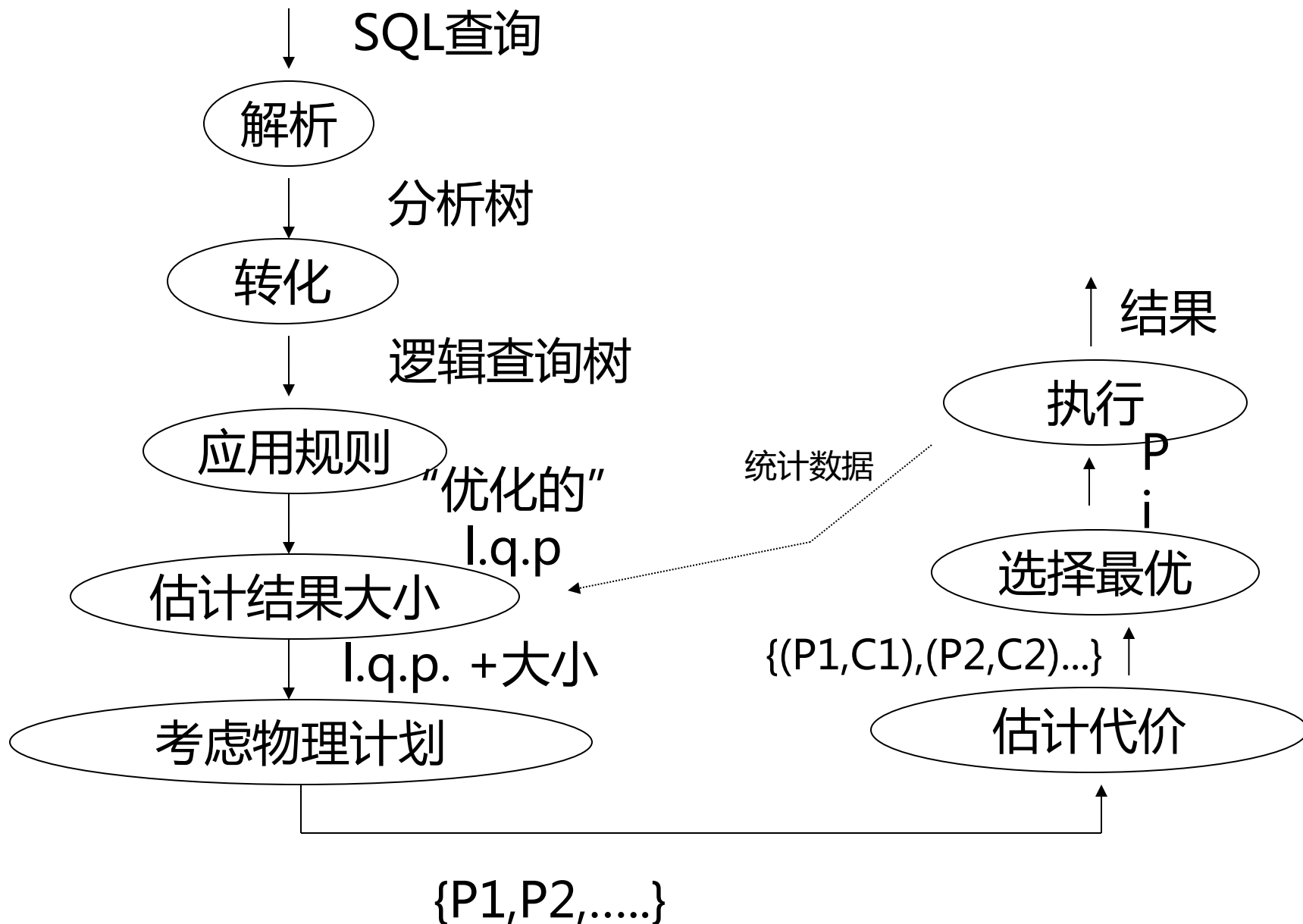


计划3



查询优化概述

查询优化概述

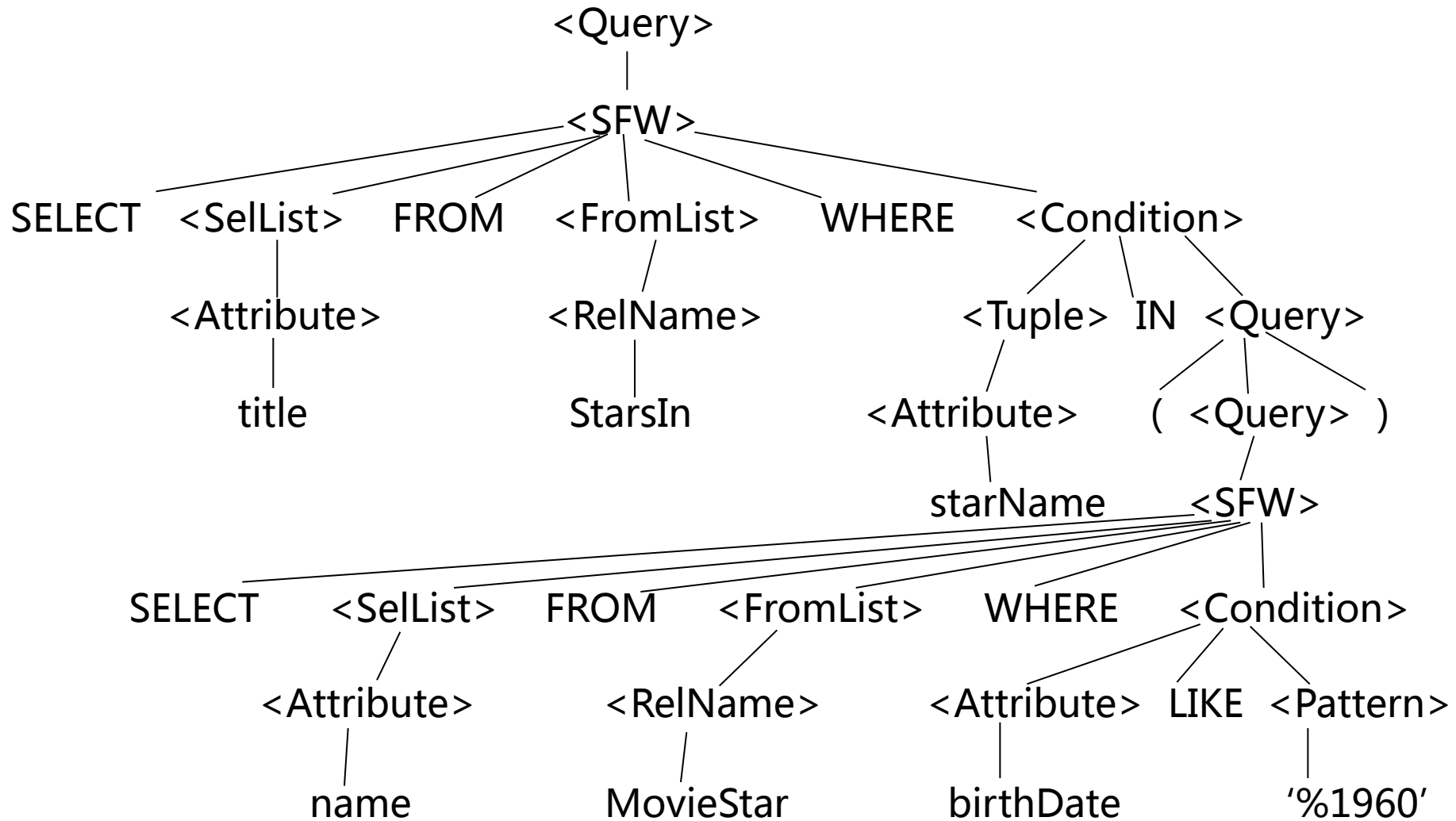


举例：SQL查询

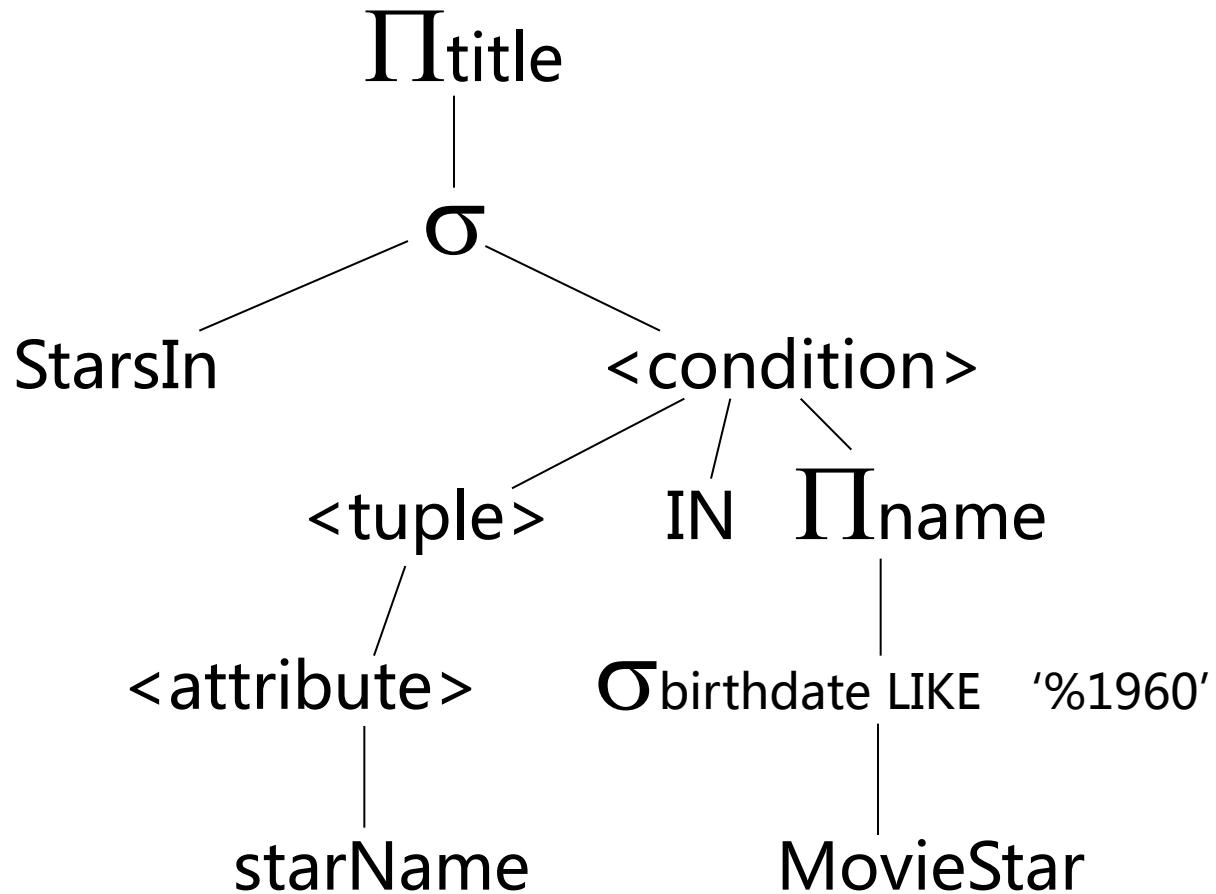
```
SELECT title
FROM StarsIn
WHERE starName IN (
    SELECT name
    FROM MovieStar
    WHERE birthdate LIKE '%1960'
);
```

(找出由1960年出生的影星出演的电影)

举例：分析树



举例：生成关系代数



举例：逻辑查询计划

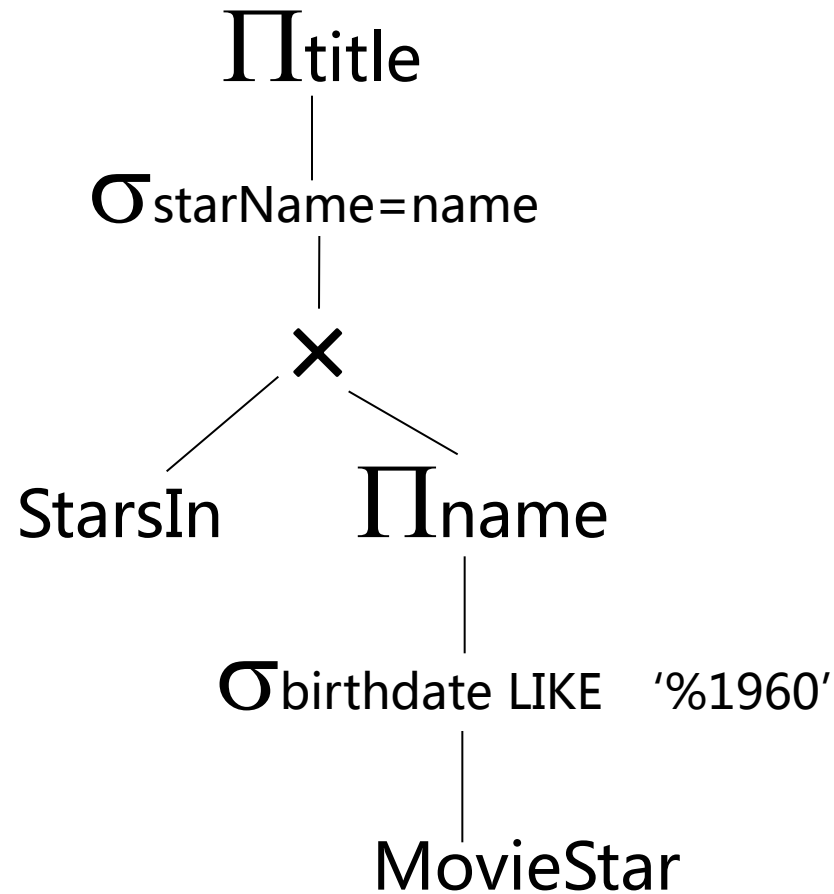
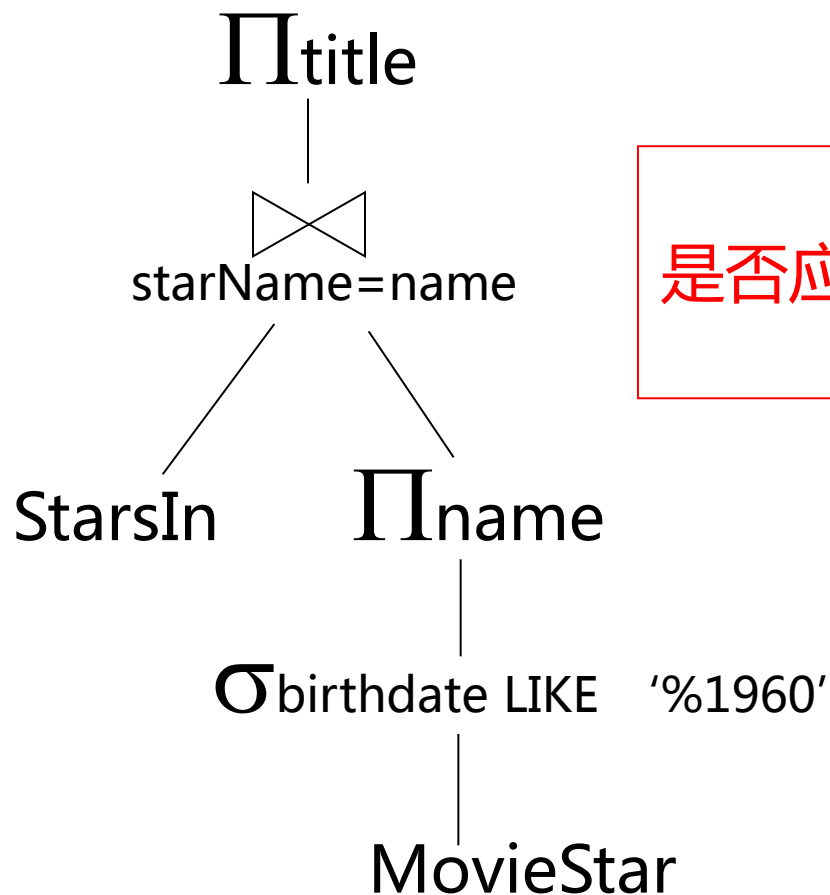


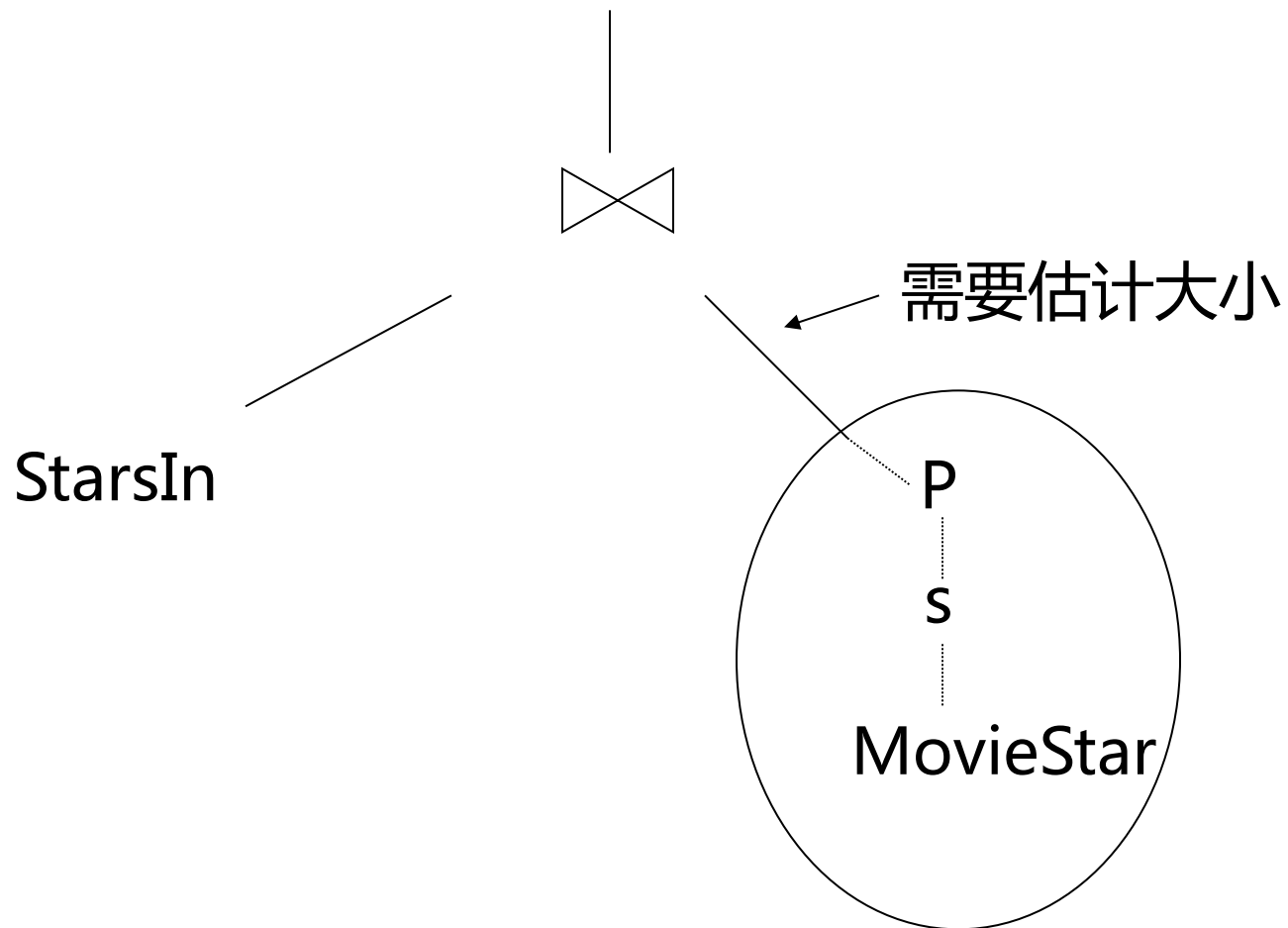
Fig. 7.18: Applying the rule for IN conditions

举例：优化后的逻辑查询计划

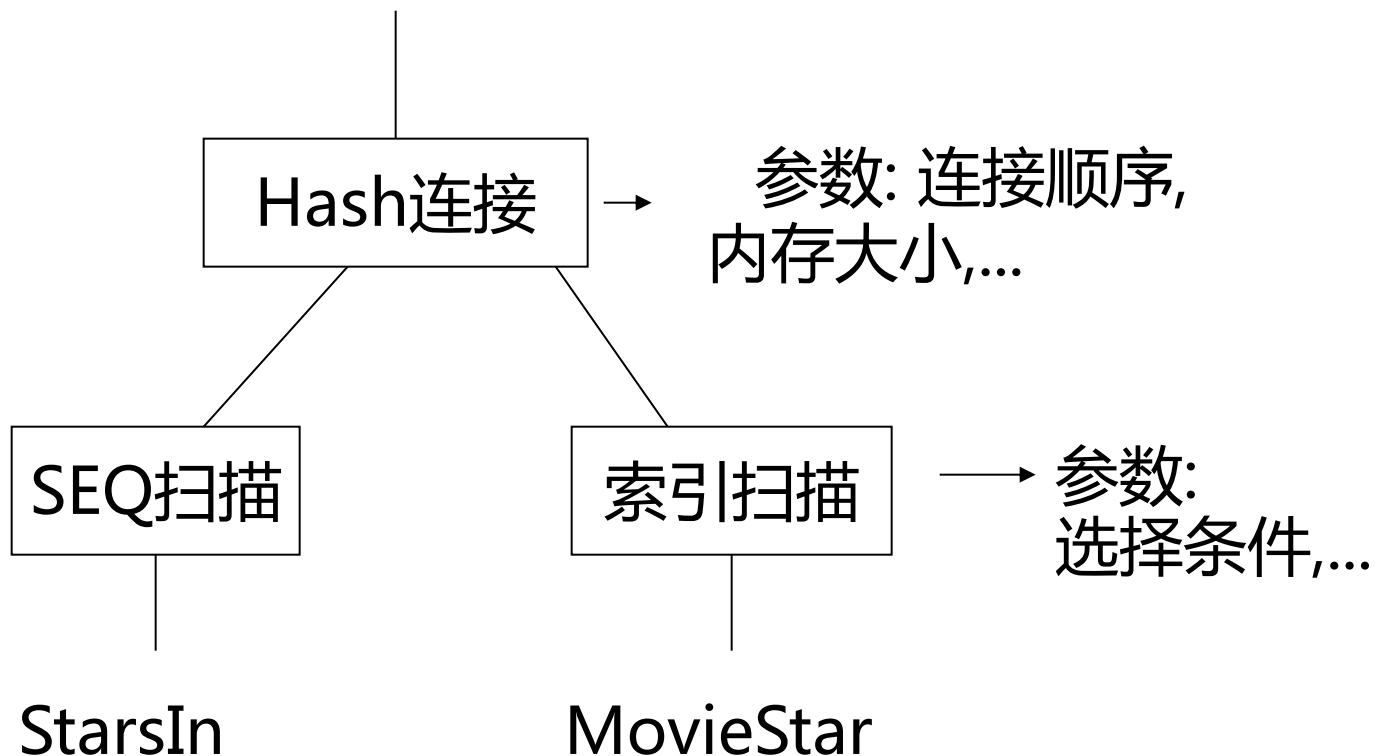


问题:
是否应当把投影下推到
StarsIn?

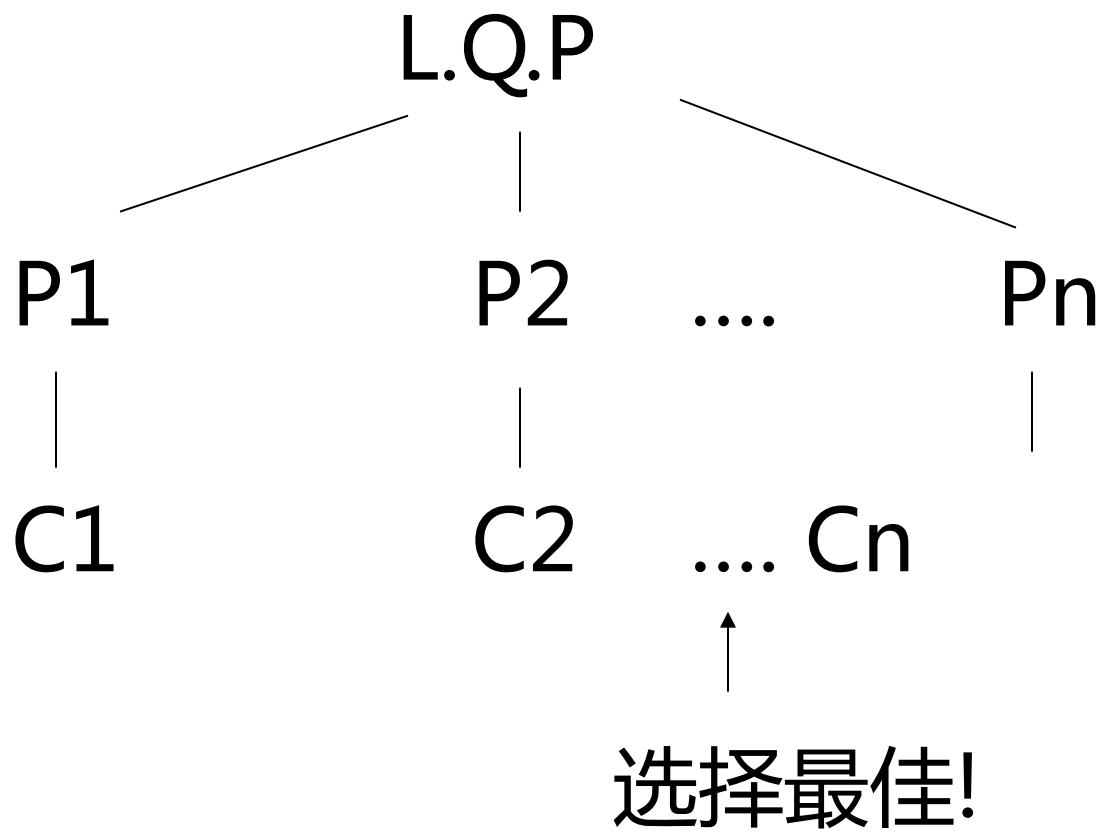
估计结果大小



举例：一个物理计划



举例：估计代价



– 事务(Transaction)的概念

- 定义：事务是DBMS中一个逻辑工作单元，通常由一组数据库的操作组成，满足原子性、一致性、隔离性和持久性四个性质。
- 事务的ACID性质
 - 原子性(Atomic)：构成事务的所有操作要么全部执行，反映到数据库中；要么全部不执行，没有对数据库中的数据造成影响。
 - 一致性(Consistency)：事务的操作使数据库从一个一致状态(所有对象满足各自的约束)转变为另一个一致状态。
 - 隔离性(Isolation)：在多用户环境下，事务的执行不受同时执行的其它事务的影响。例：多售票点售票的例子。
 - 持久性(Durability)：一个事务如果成功执行，其对数据库的影响是持久的，不因故障而失效。

– 事务的例子

- 例1：向客户1003销售pCode=101的产品10个
Update Product set qty=qty-10 where pCode= '101' ;
Insert into Order('25' , ' 1002' ,2016-3-11,2016-3-12);
Insert into Orderdetail('25' , ' 101' ,10,0.05);
Commit;
- 例2：银行数据库的转账事务，从账号A转50元到账户B
Read(A);
A:=A-50;
Write(A);
Read(B); B:=B+50;
 → Write(B);
 Commit;

– 事务的标识

- 由begin transaction语句和end transection 显式定义事务的开始和结束。
- 隐式定义：从一个读写语句开始，遇到rollback或commit语句时终止(SQL-92)。
- Commit：提交，确认事务执行。
- Rollback：回滚，将此前的执行的操作影响恢复到操作前。

– 事务的大小与回滚机制

- 事务执行过程中出现错误、或不可执行的操作(如订单已插入、结账时发现账户余额不足)需回滚。因此DBMS将事务的执行缓存在回滚段(表空间一个区域)，执行rollback或commit后清除。
- 回滚段大小、回滚缓冲区大小影响事务的大小、并发事务数和执行效率。

– 如何保证上述事务满足ACID性质？

- 回滚机制和故障恢复机制保证了事务的原子性。如果一个事务没有执行到commit，前面的操作不反映到数据库数据中。
- 只要事务定义的合理，其原子性能保证结果得一致性。
- 持久性需要数据库的故障恢复系统将发生故障时尚未将操作结果体现到数据库中的操作进行适当处理，以确保操作结果正确。
- 关于持久性，与数据库文件系统的缓存机制密切相关。因为数据的读写操作是在缓冲区进行的，在还没有写回磁盘文件前发生故障，就出现了持久性问题。
- 故障恢复机制：在事务的执行过程中，如果发生了故障，系统采取适当措施取消一个未完成的事务对数据库的影响，或将已经执行完的事务结果反映的数据库存储文件中。
- 事务的隔离性需要复杂的并发控制机制来实现。

- 并行机正成为相当普通和负担得起的机器
 - 微处理器, 内存, 磁盘的价格已经大幅下降
- 数据库正变得越来越大
 - 大量事务数据被收集和存储以用于将来的分析.
 - 数据库中存储的多媒体对象(如图像)越来越多
- 大规模并行数据库系统越来越多地用于:
 - 存储大量数据
 - 处理耗时的决策支持查询
 - 提供高吞吐量的事务处理

- 数据可划分在多个磁盘上, 导致并行I/O.
- 个别关系操作 (如排序, 连接, 合计) 可以并行执行
 - 数据可划分且每个处理器可对其划分部分独立操作.
- 查询可用高级语言(SQL, 翻译到关系代数)表达
 - 使并行化更容易.
- 不同查询可并行执行.
 - 并发控制处理冲突.
- 可见, 数据库很自然地具有并行性.

- 通过将关系划分到多个磁盘来减少从磁盘检索关系所需的时间
- 水平划分 – 关系的元组在多个磁盘间划分, 每个元组只存储在一个磁盘上.
- 划分技术 (磁盘数 = n):

循环(Round-robin)划分:

将关系中的第 i 条元组送到第 $(i \bmod n)$ 号磁盘.

散列划分:

- 选择一个或多个属性作为划分属性.
- 选择值域为 $0..n-1$ 的散列函数 h
- 令 i 表示将散列函数 h 作用于元组的划分属性值得到的结果. 则该元组送往磁盘 i .

- 划分技术(续):

- 范围划分:**

- 选择一个属性作为划分属性.
 - 选择一个划分向量 $[v_0, v_1, \dots, v_{n-2}]$.
 - 设一元组的划分属性值为 v . 使 $v_j \leq v \leq v_{j+1}$ 的元组分到磁盘 $j+1$. 使 $v < v_0$ 的元组分到磁盘0. 使 $v \geq v_{n-2}$ 的元组分到磁盘 $n-1$.

例如, 设划分向量为 $[5, 11]$, 则划分属性值等于2 的元组将分到磁盘0, 等于8 的元组将分到磁盘1, 等于20的元组分到磁盘2.

- 评估划分技术对下列类型的数据存取的支持好坏:
 1. **扫描**整个关系.
 2. 联想式定位元组 – **点查询**.
 - 例如, $r.A = 25$.
 3. 定位所有给定属性值落入指定范围的元组 – **范围查询**.
 - 例如, $10 \leq r.A < 25$.

循环划分:

- 优点
 - 最适合顺序扫描整个关系的查询.
 - 所有磁盘具有几乎相等的元组数; 因此各磁盘上的查询工作量是平衡的.
- 缺点
 - 范围查询难以处理
 - 没有聚簇(clustering) – 元组分散在所有磁盘

散列划分:

- 适合于顺序存取
 - 假设散列函数选的好, 且划分属性形成键, 则元组平均分布在各磁盘上
 - 于是各磁盘的存取工作量是平衡的.
- 适合于划分属性上的点查询
 - 可只检查单一磁盘, 使其他磁盘可用于其他查询.
 - 对划分属性的索引可局部于磁盘, 从而使查找和更新更高效
- 无聚簇, 因此难以回答范围查询

范围划分:

- 提供基于划分属性值的数据聚簇.
- 适合于顺序存取
- 适合于划分属性上的点查询: 只需存取一个磁盘.
- 对划分属性上的范围查询, 可能只需存取一个或少数几个磁盘
 - 其它磁盘可用于其他查询.
 - 当结果元组来自一个或几个块时最好.
 - 如果需存取许多块, 且仍从一个或几个磁盘取得, 则磁盘存取中潜在的并行性被浪费
 - 这是**执行偏斜**的例子.

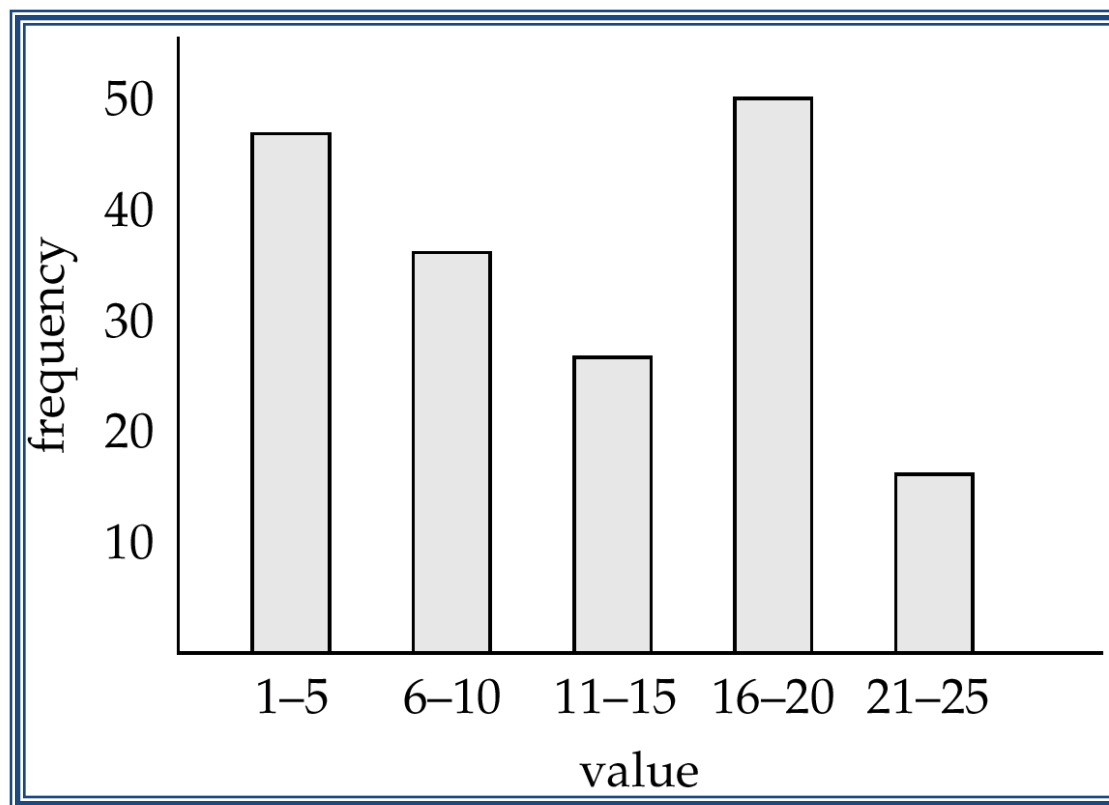
- 如果关系只包含少量元组, 可以放入单一磁盘块, 则将该关系分配到单个磁盘上.
- 大关系最好在所有可用磁盘间划分.
- 如果关系由 m 个磁盘块组成, 且系统中有 n 个可用磁盘, 则该关系应该分配 $\min(m, n)$ 个磁盘.

- 元组在磁盘间的分配可能是**偏斜的** — 即, 某些磁盘上有许多元组, 而另一些只有少量元组.
- **偏斜的种类:**
 - **属性值偏斜**
 - 某些值在许多元组的划分属性上出现; 所有在划分属性上值相同的元组被分配在同一分区中.
 - 范围划分和散列划分中可能发生.
 - **划分偏斜**
 - 对范围划分, 一个坏的划分向量可能将过多元组分配到一个分区以及过少元组分配到其他分区.
 - 对散列划分, 只要选择好的散列函数就不太可能发生.

- 为生成平衡的划分向量(假设划分属性是关系的键):
 - 对关系按划分属性排序.
 - 按序扫描关系并构造划分向量如下.
 - 每读出关系的 $1/n$, 下一条元组的划分属性值就加入划分向量.
 - n 表示将构造的分区数.
 - 如果划分属性有重复值则可导致重复项或不平衡.
- 实际中还使用基于直方图的其他技术

利用直方图处理偏斜

- 从直方图可以相对直接地构造出平衡的划分向量
 - 假设在直方图的每个范围中一致分布
- 直方图可通过扫描关系或者对关系元组抽样来构造



利用虚拟处理器来处理偏斜

- 范围划分中的偏斜可以用**虚拟处理器**很好地处理:
 - 创建大量虚拟处理器(比如是实际处理器数目的10 到20倍)
 - 用任何划分技术将关系元组分配到虚拟处理器
 - 以循环方式将每个虚拟处理器映射到实际处理器
- 基本思想:
 - 如果正常划分会导致偏斜, 则偏斜很可能被分散到若干个虚拟分区
 - 偏斜的虚拟分区被分散到若干实际处理器上, 于是分配变得平均了!

查询间并行

- 查询/事务相互并行执行
- 增加事务吞吐量; 主要用于扩展(scale up)事务处理系统以支持更大的每秒事务数.
- 最容易支持的并行形式, 特别是在共享内存并行数据库中, 因为即使串行数据库系统也支持并发处理.
- 在共享磁盘或无共享体系(share-nothing)结构上的实现更复杂
 - 必须在处理器之间传送消息来协调封锁和日志登记.
 - 本地缓冲区中的数据可能已被另一处理器更新.
 - **缓存一致性(cache-coherency)**得到保持 — 对缓冲区数据的读写必须找到数据的最近版本.

- 共享磁盘系统的缓存一致性协议例:
 - 读/写一页之前, 该页必须以共享/排他方式加锁.
 - 对页加锁时, 该页必须从磁盘读出
 - 释放页锁之前, 该页如果更新过则必须写到磁盘.
- 存在更复杂但磁盘读写次数较少的协议.
- 无共享系统的缓存一致性协议是类似的. 每个数据库页都有一个 *home* 处理器. 对页的读写请求都发往它的 *home* 处理器.

- 单个查询在多个处理器/磁盘上并行执行; 对加速耗时查询很重要.
- 查询内并行的两种互相补充的形式:
 - **操作内并行** – 查询内每个操作并行执行.
 - **操作间并行** – 查询内不同操作并行执行.

第一种形式在增加并行度时的伸缩性好, 因为每个操作处理的元组数通常都多于查询内的操作数目

- 我们讨论并行算法时假定:
 - 查询是只读的
 - 无共享体系结构
 - n 个处理器 P_0, \dots, P_{n-1} , 及 n 个磁盘 D_0, \dots, D_{n-1} , 其中磁盘 D_i 与处理器 P_i 关联.
- 如果一个处理器有多个磁盘, 可以简单地模拟成单个磁盘.
- 无共享体系结构可以在共享内存和共享磁盘系统上有效地模拟.
 - 因此无共享系统的算法可在共享内存或共享磁盘系统上执行.
 - 但是, 可能需要一些优化.

范围划分排序

- 选择处理器 P_0, \dots, P_m 执行排序, 其中 $m \leq n - 1$.
- 在排序属性上创建有 m 个项的范围划分向量
- 利用范围划分重新分配关系
 - 落在第 i 个范围内的所有元组送往处理器 P_i
 - P_i 将接收到的元组临时存储在磁盘 D_i
 - 这一步需要I/O 和通信开销.
- 各处理器 P_i 对本地的关系分片进行排序.
 - 各处理器并行执行同样的操作(排序), 相互没有交互 (**数据并行**).
- 归并结果正确性: 范围划分确保对 $i < j \leq m$, 处理器 P_i 上的键值都小于 P_j 上的键值.

并行外排序-归并

- 假设关系已经划分到磁盘 D_0, \dots, D_{n-1} (任何划分方法).
- 各处理器 P_i 对磁盘 D_i 上的数据进行本地排序.
- 各处理器上排好的序段再进行归并以得到最终的排序输出.
- 序段的归并可并行化, 如下:
 - 将各处理器 P_i 上的有序分片范围划分到处理器 P_0, \dots, P_{m-1} .
 - 各处理器 P_i 对接收到的输入流进行归并, 得到单一序段.
 - 处理器 P_0, \dots, P_{m-1} 上的各序段合并即得到最终结果.

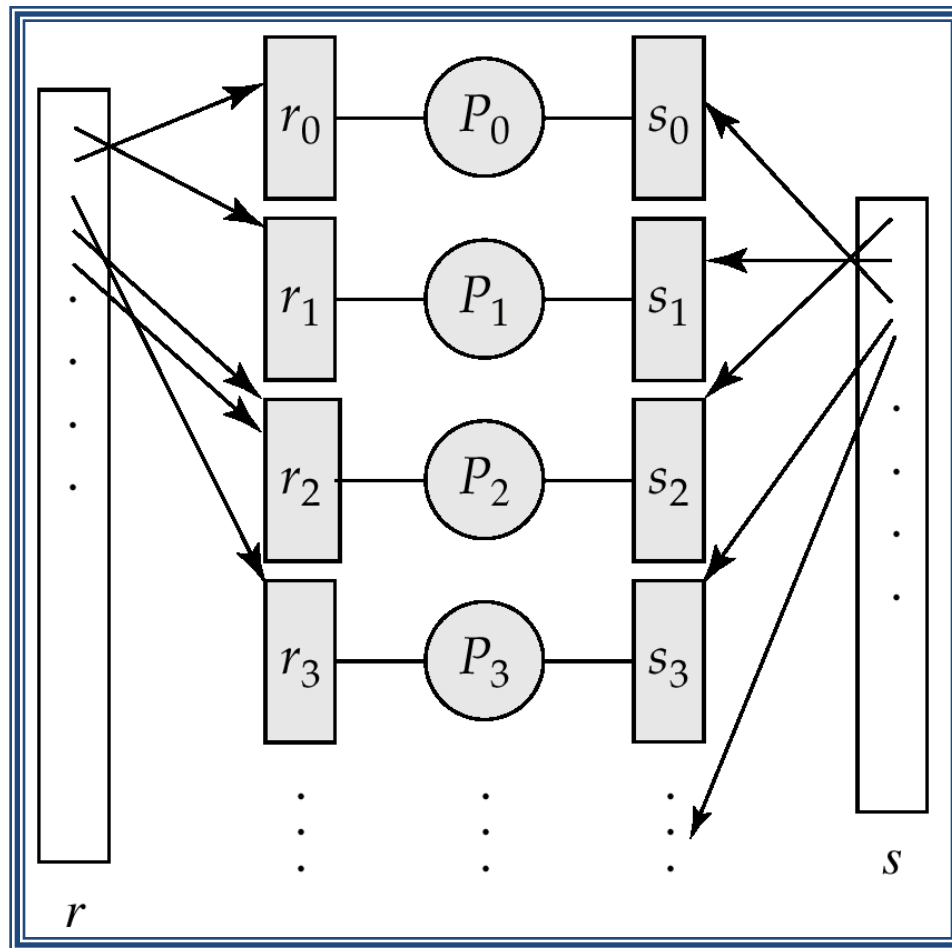
并行连接 (join)

- 连接操作需要测试成对元组看它们是否满足连接条件, 如果满足则该元组对加入连接输出结果.
- 并行连接算法试图将待测试元组对划分到若干个处理器上, 每个处理器在本地计算部分连接结果.
- 最后, 各处理器上的结果汇集成为最终结果.

划分连接

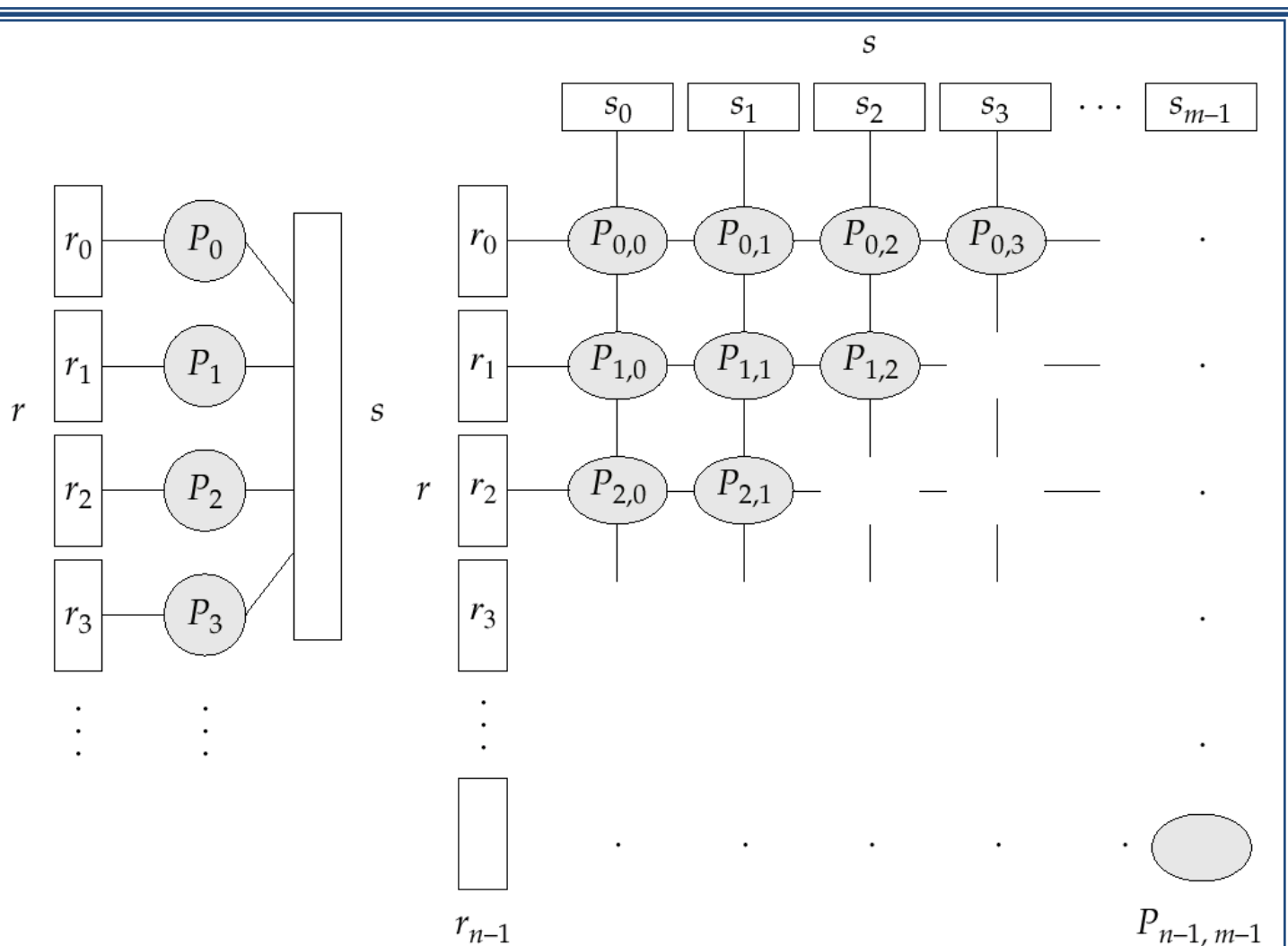
- 对等值连接和自然连接, 可以将两个输入关系划分到多个处理器上, 各处理器在本地计算连接.
- 令 r 和 s 是输入关系, 我们要计算 $r \bowtie_{r.A=s.B} s$.
- r 和 s 分别划分成 n 个分片, 记为 r_0, r_1, \dots, r_{n-1} 和 s_0, s_1, \dots, s_{n-1} .
 - 可以用范围划分或散列划分.
 - r 和 s 必须根据其连接属性 $r.A$ 和 $s.B$ 来划分, 并使用相同的范围划分向量或散列函数.
 - 分片 r_i 和 s_i 分配到处理器 P_i .
- 各处理器 P_i 在本地计算 $r_i \bowtie_{r_i.A=s_i.B} s_i$. 可以使用任何标准连接方法.

划分连接



- 对某些连接条件不能用划分方法
 - 例如, 非等值连接条件, 如 $r.A > s.B$.
- 对于无法应用划分的连接, 可以使用**分片与复制**技术来达到并行化
 - 下一页图示
- 特例 – **非对称分片与复制**:
 - 一个关系进行划分, 设为 r ; 可用任何划分技术.
 - 另一个关系, 设为 s , 则在所有处理器上复制.
 - 处理器 P_i 在本地计算 r_i 与 s 的连接, 可用任何连接技术.

分片与复制连接



a. 非对称分片与复制

b. 分片与复制

分片与复制连接

- 一般情形: 减小各处理器上的关系的大小.
 - r 划分成 n 个分片 r_0, r_1, \dots, r_{n-1} ; s 划分成 m 个分片 s_0, s_1, \dots, s_{m-1} .
 - 可用任何划分技术.
 - 至少应有 $m * n$ 个处理器.
 - 记为 $P_{0,0}, P_{0,1}, \dots, P_{0,m-1}, P_{1,0}, \dots, P_{n-1,m-1}$.
 - P_{ij} 计算 r_i 与 s_j 的连接. 为此, r_i 被复制到 $P_{i,0}, P_{i,1}, \dots, P_{i,m-1}$, 而 s_i 被复制到 $P_{0,i}, P_{1,i}, \dots, P_{n-1,i}$.
 - 可用任何连接技术

分片与复制连接

- 分片与复制的两个版本都可在任何连接条件下运行, 因为 r 的每个元组可与 s 的每个元组测试连接条件.
- 通常比划分连接代价高, 因为一个关系(对非对称分片与复制)或两个关系 (对一般分片与复制)必须复制.
- 即使可以使用划分技术, 有时非对称分片与复制也更可取.
 - 例如, 若 s 小而 r 很大, 且已作划分. 可能更廉价的连接方法是将 s 复制到所有处理器, 而不是将 r 和 s 重新在连接属性上划分.

并行嵌套循环连接

- 假定
 - 关系 s 远远小于关系 r , 且已划分存储.
 - 在关系 r 的每个分片上, r 的连接属性上存在一个索引.
- 将关系 s 复制, 利用现有的关系 r 分片, 采用非对称分片与复制.
- 存有关系 s 分片的各处理器 P_j 读入存储在 D_j 上的关系 s 的元组, 并复制到其他各处理器 P_i .
 - 本阶段结束时, 关系 s 在所有存有关系 r 的元组的机器上得到复制.
- 各处理器 P_i 执行关系 s 与关系 r 的第 i 个分片的索引嵌套循环连接.

选择 $\sigma_{\theta}(r)$

- 若 θ 形如 $a_i = v$, 其中 a_i 是属性而 v 是值.
 - 若 r 在 a_i 上做了划分则选择在单个处理器上进行.
- 若 θ 形如 $l \leq a_i \leq u$ (即 θ 是范围选择) 且关系在 a_i 上做了范围划分
 - 选择在其划分与指定值范围有重叠的各个处理器上执行.
- 其他所有情形: 选择在所有处理器上并行执行.

- 复本删除
 - 利用任一并行排序技术执行
 - 排序过程中一旦发现复本就删除之.
 - 也可以划分元组(利用范围或散列划分) 并在每个处理器本地执行复本删除.
- 投影
 - 不带复本删除的投影可以在从磁盘并行读入元组时进行.
 - 如果需要复本删除, 以上任何副本删除技术都可用.

- 将关系按分组属性划分然后在每个处理器本地计算聚合值.
- 通过在划分前部分计算聚合值可以减少划分时传送元组的代价.
- 考虑**sum**聚合运算:
 - 在每个处理器 P_i 对存储在磁盘 D_i 上的元组执行聚合运算
 - 导致在每个处理器上有部分和的元组.
 - 局部聚合的结果按分组属性划分, 然后在每个处理器 P_i 上再次执行聚合以得到最终结果.
- 划分时只需将较少元组送到其他处理器.

• 流水线并行

- 考虑四个关系的连接: $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$
- 建立一个流水线来并行计算三个连接
 - 令P1 计算 $\text{temp1} = r_1 \bowtie r_2$
 - 令P2 计算 $\text{temp2} = \text{temp1} \bowtie r_3$
 - 令P3 计算 $\text{temp2} \bowtie r_4$
- 这些操作可以并行执行: 在计算进一步结果的同时可以将已计算出的结果元组送到下一操作
 - 需要用可流水线化的连接算法 (如索引嵌套循环连接)

限制流水线并行可用性的因素

- 流水线并行有用是因为它避免了将中间结果写到磁盘
- 对少量处理器有用, 但对较多处理器伸缩性不好. 一个原因是流水线链不能达到足够的长度.
- 对于需要取得所有输入后才能产生输出的操作(如合计与排序)不能流水线化
- 对于经常发生的偏斜情形, 一个操作的执行代价远远高于其他操作, 这时只能获得很少的加速比.

- **独立并行**

- 考虑四个关系的连接

$$r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$$

- 令P1 用来计算 $\text{temp1} = r_1 \bowtie r_2$
 - P2 用来计算 $\text{temp2} = r_3 \bowtie r_4$
 - P3 用来计算 $\text{temp1} \bowtie \text{temp2}$
 - P1 和 P2 可以独立地并行工作
 - P3 必须等待来自P1 和P2的输入
 - 可以将P1与P2输出到P3的过程流水线化, 从而结合了独立并行和流水线并行

- 并没有提供高度的并行

- 对较低程度的并行有用.
 - 在高度并行系统中用途不大

查询优化

- 并行数据库中的查询优化比串行数据库中的查询优化复杂得多.
- 代价模型更复杂, 因为我们必须考虑划分代价以及诸如偏斜和资源竞争等问题.
- 在并行系统中**调度**执行树时, 必须决定:
 - 如何并行化每个操作以及为它要用多少处理器.
 - 哪些操作该流水线化, 哪些操作该并行地独立执行, 哪些操作该顺序执行.
- 为每个操作确定所分配资源的数量是个问题.
 - 例如, 分配比最优方案更多的处理器可导致很高的通信开销.
- 应该避免长流水线, 因为最后一个操作可能为输入等待很久, 同时占有宝贵资源

- 要从中进行选择并行执行计划数目比顺序执行计划的数目大得多。
 - 因此优化时需要启发式
- 选择并行计划的两种可替换启发式:
 - 不用流水线和操作间流水线; 只要在所有处理器间并行化每个操作.
 - 找到最佳计划现在就容易多了 --- 使用标准优化技术, 但是依据新的代价模型
 - 首先选择最高效的顺序计划, 然后选择如何最好地将该计划中的操作并行化.
 - Volcano 并行数据库使**交换操作符**模型流行起来
 - 在查询计划中引进交换操作符以划分和分布元组
 - 在每个处理器上, 每个操作在本地数据上独立工作, 与该操作的其他拷贝并行进行
 - 作为一个选择可以探索流水线并行
- 选择好的物理组织(划分技术)对加速查询很重要.

● 数据规模增大

- 需要面向大数据的数据库管理系统具有处理大规模数据的能力，确保大规模数据能够“存得下，查得出”，有力支撑更加复杂的操作
- 不可避免的使用了分布式系统

➤ 数据类型多样

- 大数据经常包含着结构化、半结构化以及非结构化的数据
- 要求数据库管理系统能够适应结构化、半结构化以及结构化数据

大数据对数据管理的新需求

● 模式数据关系演化

- 传统的数据库先有模式，后于模式数据录入数据
- 大数据时代难以预先确定固定模式，模式只有在数据出现之后才能确定，且不断变化

● 数据作用复杂

- 传统数据库的数据仅为处理对象；而对于大数据，数据作为一种资源来辅助解决其他诸多领域的问题
- 要求数据库管理系统系统中有更复杂的数据操作并且提供更多工具

● 处理工具专门化

- One size fits all (关系数据库)→No size fits all (NoSQL, NewSQL)