厦门大学计算机科学系研究生课程

《分布式数据库技术》

专题三 分布式查询处理与优化

（2012年新版）

林子雨

厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

主页：**http://www.cs.xmu.edu.cn/linziyu**

第4章 分布式查询处理与优

4.1 分布式查询概述
4.2 分布式查询优化的基础知识
4.3 全局查询到逻辑查询的转换
4.4 逻辑查询到物理查询的转换
4.5 基于半联接的查询优化
4.6 基于枚举法的查询优化

4.1 分布式查询概述

4.1.1 分布式查询处理的定义与分类

- 4.1.1.1 集中式数据库查询的基本原理
- 4.1.1.2 分布式查询处理中的三种查询
- 4.1.1.3 分布式查询定义
- 4.1.1.4 分布式查询的分类

4.1.2 分布式查询优化的目标

4.1.3 分布式查询优化的准则和代价分析

4.1.4 分布式查询优化策略的重要性

4.1.1.1 集中式数据库查询的基本原理

在集中式数据库环境中的查询处理，是将用户查询（由查询语言表达）转换成物理查询处理，其中包括了物理优化和逻辑优化两个次。

物理优化：对关系（数据库）的基本操作符的运算在实现中的优化（如索引、排序、聚集（聚簇）等）

逻辑优化：在进行物理优化前先应有一个合理的（最优的）操作符次序或一些操作策略的选择

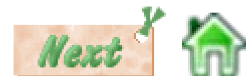
4.1.1.2 分布式查询处理中的三种查询

分布式数据库环境是由虚拟的全局数据库和实际存在的各局部数据库组成的。**DDB**的分布性，使虚拟的全局数据库在抽象时还有逻辑数据库（**LgDB)**和物理数据库（**PDB**）的概念；

DDB的四层模式结构中的局部概念层是由物理数据库映射到局部场地上的，即形成局部数据库；

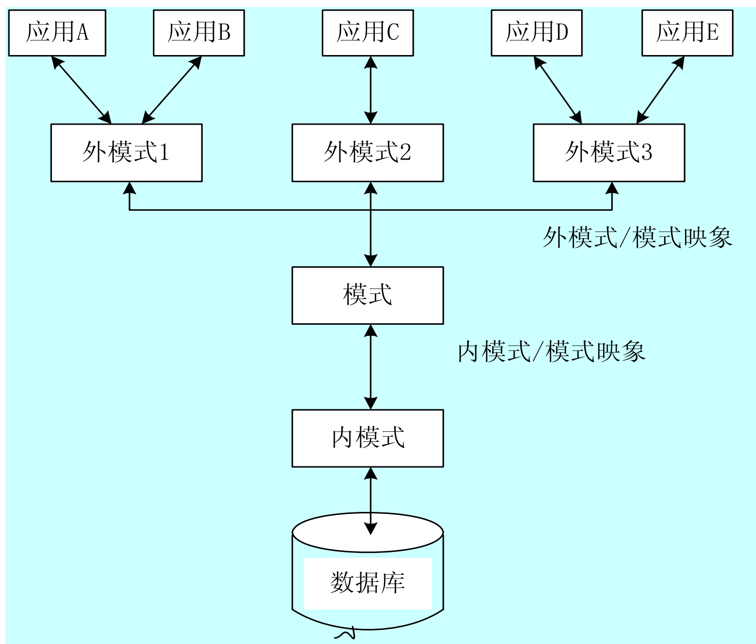
分布式查询处理包含了全局查询处理和局部查询处理两个部分。但是，对使用 **DDB** 来说，用户（应用）只看到 **GDB**，且也只在全局关系上执行查询。而用户的这种查询是通过查询语言表示，并由系统将其转换。因而在查询执行过程中，实际上最终要涉及到具体场地上的物理关系的查询。

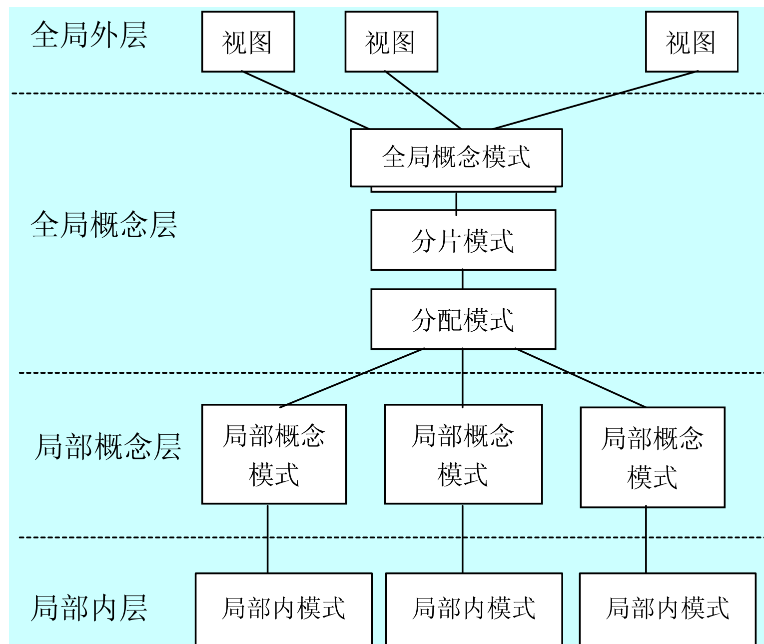
因此，分布式查询对应全局层三种数据库有三种查询：**用户查询、逻辑查询和物理查询**。



4.1.1.2 分布式查询处理中的三种查询

集中式三层摸式结构图





分布式四层摸式结构图

4.1.1.2 分布式查询处理中的三种查询

用户查询（Q_u）：

是DDB中在全局数据库（GDB）上的查询；

逻辑查询（Q_L）：

是DDB中在逻辑数据库（LgDB）上的查询；

物理查询（Q_p）：

是DDB中在物理数据库（PDB）上的查询。

4.1.1.2 分布式查询处理中的三种查询

上述三种查询之间有一定的联系，这种联系依赖于DDB的分片模式定义和分配模式定义，我们可用以下定理来描述：

［ 定理 4.1 ］：对于任一用户查询Q_u，

相应的逻辑查询为$Q_L = Q_u \cdot FS^{-1}$，

相应的物理查询为$Q_P = Q_u \cdot FS^{-1} \cdot AS^{-1}$。

证明：由3.1节分片模式定义，有$GDB=FS^{-1}(LgDB)$，

所以，有$Q_u(GDB)= Q_u(FS^{-1}(LgDB))= Q_u\ FS^{-1}(LgDB)$；

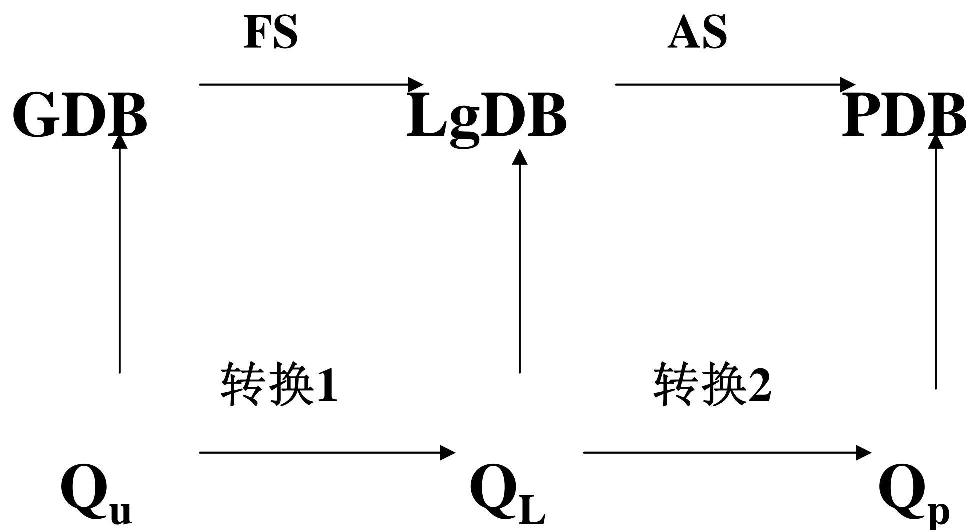
同样，由3.1节分配模式定义，有$LgDB=AS^{-1}(PDB)$；

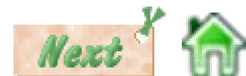
所以有 $GDB=FS^{-1}(LgDB)=FS^{-1}\cdot AS^{-1}(PDB)$，

因而，有$Q_u(GDB)= Q_u(FS^{-1}\cdot AS^{-1}(PDB))= Q_u.FS^{-1}\cdot AS^{-1}(PDB)$ 。

4.1.1.3 分布式查询定义

定理4.1讨论的是全局层的查询处理，其中对用户查询，在系统中实际存在了两次转换：全局查询到逻辑查询的转换和逻辑查询到物理查询的转换。如下图所示：





4.1.1.3 分布式查询定义

[定义4.1]：分布式数据库系统的查询处理Q是一算法：算法的输入是用查询Q_u，算法的输出是相应的物理查询Q_p；算法的功能是将用户查询按每个全局关系的分布结构转换成一个最优的物理查询。

DDB查询处理和优化主要讨论以下问题：

- 全局查询处理的转换、优化
- 由于分布性引起的数据在场地间移动时的数据副本选择和查询操作序的确定等策略

对于物理查询的具体执行，就相当于在一个集中式数据库上的操作（即对局部数据库的操作），其上的查询处理属于局部查询处理。集中式数据库所讨论的查询处理及优化策略是本专题的技术基础。

4.1.1.4 分布式查询的分类

- 在分布式环境下，查询可以分为三类：
- 局部查询
  - 在本站点上执行查询，即查询本站点上的数据
  - 可以使用集中式查询处理技术
- 远程查询
  - 在某个站点上执行查询，即查询在网络上的另一个站点上存放的数据
- 全局查询
  - 涉及远程站点或多个站点上的数据，必须进行站点间的通信

4.1.1.4 分布式查询的分类

（1）局部查询

- 局部查询只涉及本地、单个站点上的数据，所以，查询优化技术与集中式数据库相同，如分解查询、关系代数表达式的等价转换和基于代价的估算等

局部查询的优化策略包括：

- 选择和投影运算尽可能先做，使得运算中间结果数据量大大减少
- 在执行连接前对数据库进行适当的预处理，如对数据按连接属性排序，和在连接属性上建立索引，以减少扫描次数，提高连接速度
- 同时执行一串连接投影和选择操作，且尽可能把它们与其前后的二元操作结合起来，避免重复扫描关系和减少中间数据，如把笛卡尔积与其后的选择操作合并成自然连接
- 找出公共子表达式

4.1.1.4 分布式查询的分类

（2）远程查询

- 远程查询只涉及单个站点的数据，所以，它的局部处理的优化策略与本地查询所采取的优化策略相同
- 远程查询还涉及远程站点的选择，即选择哪个远程站点上的数据或数据片段作为查询的对象
- 为了减少远程查询的通信代价，如果数据是冗余分配，应尽可能选择距离发出查询应用的站点最近的站点上的数据或数据片段作为查询对象

4.1.1.4 分布式查询的分类

（3）全局查询

- 涉及多个站点上的数据，因此，查询处理和优化技术要复杂得多
- 为了执行全局查询并确定一个好的查询策略，需要做很多判断和计算工作，具体包括：
  - 具体化：确定查询使用的物理副本，落实查询对象
  - 确定操作执行的次序：主要是确定二元连接和并操作的次序，其他操作的次序是不难确定的。
  - 确定操作执行的方法：把若干个操作连接起来在一次数据库访问中执行，确定可用的访问路径（比如索引）。
  - 确定执行站点：全局查询执行站点，并不一定就要在查询发出的站点，只要把最终结果送回始发站点即可。不同站点执行代价不同，需要综合考虑。

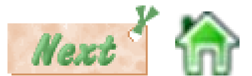
4.1.2 分布式查询优化的目标

- 以总代价最小为目标
  - 除了和集中式数据库一样考虑CPU代价和I/O代价之外，总代价还包括通过网络在站点之间传输数据或信息的代价。
- 以每个查询的响应时间最短为目标
  - 数据的分布和冗余，增加了查询的并行性，缩减响应时间
- 查询优化目标的选择
  - 可以同时使用两种标准，根据系统应用特点的不同，一种作为主要标准，一种作为辅助标准。例如，先找到一个总代价最小的方案，然后，在总代价不增加的情况下，不断修正方案，使得响应时间尽可能短。

4.1.3 分布式查询优化的准则和代价分析

(1) 查询优化准则

- 分布式查询优化的准则是使通信费用最低和响应时间最短，即以最小的总代价，在最短的响应时间内获得需要的数据。
- 所谓响应时间，是从接收查询到完成查询所需要的时间，它即与通信时间有关，也与局部处理时间有关。
- 通信费用与所传输的数据量及通信次数成正比，也与所处的通信网络类型有关，对于不同网络类型，有着不同的查询优化方法：
- 在远程通信网络中
  - 数据传输速度比内存和磁盘的数据传输速度慢许多，因此，查询的局部处理时间和通信时间相比，可以忽略不计，减少通信费用称为分布查询优化的主要目标
- 在高速局域网络中
  - 数据传输时间要比局部处理时间短许多，减少局部处理时间是优化目标



4.1.3 分布式查询优化的准则和代价分析

(2) 分布式查询的代价因素

- I/O代价（即估算输入/输出操作次数）
- CPU的使用情况
- 传输代价（包括数据量的传输费用、传输的延迟时间，以及涉及传输数据的控制信息及控制次数）
- 分布事务处理的管理策略（事务可串行化、分布式并发控制、分布式恢复）
- 分布查询操作方法（如联接操作、并操作、二元操作以及全局查询和局部查询的不同操作）对效率的影响

4.1.4 分布式查询优化策略的重要性

例：在教学数据库中，有如下表：

S(S#,SNAME,AGE,SEX) 有10^4个元组，在站点A存放

C(C#,CNAME,TEACHER) 有10^5个元组，在站点B存放

SC(S#,C#,GRADE)有10^6个元组，在站点A存放

假定每个元组的长度均为100bit，通信系统的传输速度为10^4 bit/s，通信延迟为1s

问题：要求查询所有选修"MATHS"课程的男学生的学号和姓名。

解：在分片透明性的DDBMS支持下，SQL语句是：

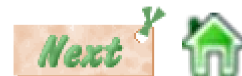
SELECT S#,SNAME FROM S,SC,C

WHERE S.S#=SC.S# AND SC.C#=C.C# AND SEX='M' AND CNAME='MATHS'

通信代价的估算公式是：

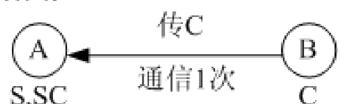
T=传输延迟时间C0+（传输的数据量X*数据传输速率C1）

=（传输次数*1）+（传输的bit数/10^4）



4.1.4 分布式查询优化策略的重要性

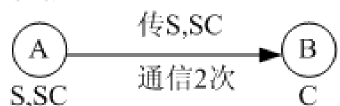
策略1



把关系C传输到A地，在A地查询处理

T1=1+（10^5*100/10^4）=10^3秒=16.7分钟

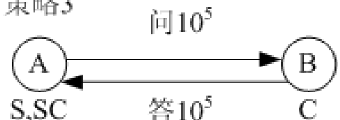
策略2



把关系S和SC传输到B地，在B地查询处理

T2=2+(10^4+10^6）*100/10^4=10102秒=28小时

策略3

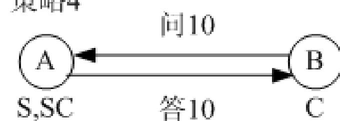


先在A地求出男学生的成绩元组，假设有10^5个

再根据C#的值询问B地，核实是否CNAME=’MATHS’

T3=(2*10^5*1)=2*10^5秒=2.3天

策略4

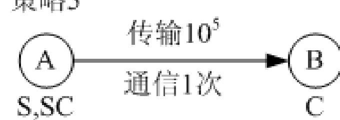


先在B地求出CNAME=’MATHS’的元组，假设有10个

再根据C#的值询问A地的S和SC的连接，核实是否为选修’MATHS’的男生

T4=(2*10*1)=20秒

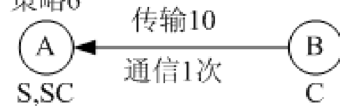
策略5



先在A地求出男生选课元组，假设有10^5个，再把结果传输到B地，在B地执行查询

T5=1+（10^5*100）/10^4=1000秒=16.7分钟

策略6



先在B地求出CNAME=’MATHS’的元组，假设有10个

再把结果传输到A地，在A地执行查询

T6=1+(10*100)/10^4=1秒

4.1.4 分布式查询策略的重要性

表　不同查询策略结果的比

处理策略	方法	时间
1	把关系C送到A地，在A地进行查询处理	16.7分钟
2	把关系S和SC传送到B地，在B地查询处理	28小时
3	对每个男生的成绩核查相应的课程名	2.3天
4	取课程名为'MATHS'的课程号，核查为男生的记录	20秒
5	把男生记录送到B地，在B地查询处理	16.7分钟
6	把课程名为'MATHS'的记录送到A地，在A地查询处理	1秒

4.2 全局查询转换的基础知识

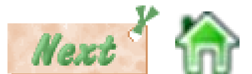
4.2.1 查询表达式的等价性

4.2.2 查询树

4.2.3 等价变换规则

4.2.1 查询表达式的等价性

- 关系数据模型有三种查询语言：代数语言、元组演算语言、域演算语言，这三种语言是等价的
- 用代数语言表达查询处理最为方便
- **SQL** 语言是关系数据库的标准语言，它是完备的，对用户而提供非过程的查询语言最为方便
- 这里假设 **DDBMS** 提供完全透明，全局用户可以用**SQL**语言操纵语句来表达全局查询，**SQL**语句是对**DDB**进行查询的外部表达式



4.2.1 查询表达式的等价性

- 查询要得到结果，必须对关系进行具体操作：
  - 五种基本操作

  并（∪）、差（-）、迪卡尔积（×）、选择（σ）、投影（π）
  - 五种导出操作

  交（∩）、商（÷）、联接（$\infty_{i\theta j}$）、自然联接（∞）、半联接（∝）
- 为了便于查询处理的转换，将上面的十种关系操作数分为两类：
  - 一元操作，用**U**表示
  - 二元操作，用**B**表示

  属于一元操作的只有σ和π，而其余的操作都是二元操作

4.2.1 查询表达式的等价性

[例]：对全局关系 **Emp**,有如下**SQL**查询表达式

```
Select   ENAME,DNO
From     Emp
Where    DNO='15';
```

（4-1）

其相应的代数操作表达式为：

$\pi_{ENAME, DNO}(\sigma_{DNO='15'} Emp)$ （4-2）

或

$\sigma_{DNO='15'}(\pi_{ENAME, DNO} Emp)$ （4-3）

本例表示了不同的操作顺序仍可获得相同的结果。这就是等价的概念。

[定义4.2]：两个查询表达式 **E1** 和 **E2** 是等价的，如果其查询的结果是相同的，记为 **E1 ≡ E2**。

4.2.2 查询树

[定义4. 3]： 查询树是一棵树 **T=(V,E)**，其中：

1）**V**是节点集，每个非叶结点是关系操作符，叶节点是关系名（即查询涉及的关系）；

2）**E**是边集，二节点有边（V_1,V_2），当且仅当 V_2 是 V_1 的操作分量。

通常，人们用查询树表示查询表达式的内部结构。

4.2.2 查询树

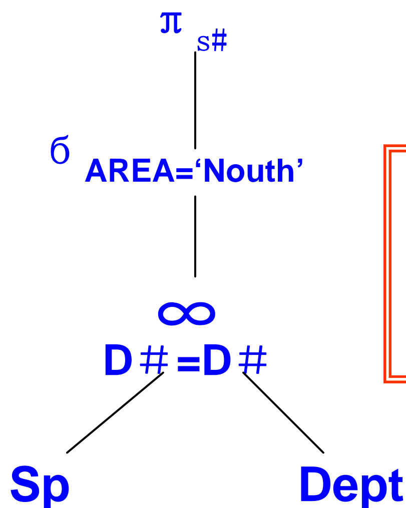
例4.1：有查询Q_1：查询北部地区（AREA='North'）所属部门发出订单的供应商号。这里涉及两个全局关系Dept(D#,DNAME,MGTRSSN,AREA)和Sp(S#,P#,D#,QUAN)，SQL表达式为：

```
Select  S#
From  Dept, Sp
Where  Sp•D#=Dept.•D#  And  AREA='North';
```

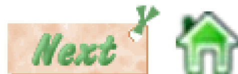
（复习多表连接内容）其相应的代数表达式为：

$$\pi_{S\#}\,\sigma_{AREA='North'}\,(Sp \underset{D\#=D\#}{\infty} Dept)$$

其相应的查询树如下：



显然，边为 E_1（$\underset{D\#=D\#}{\infty}$，Sp）时，则Sp是非叶节点 ∞ 的分量。



4.2.2 查询树

例子：多表连接操

表Student，存放学生基本信息

StudentID	StudentName	StudentAge
1	张三	25
2	李四	26
3	王五	27
4	赵六	28
5	无名氏	27

表BorrowBook，存放学生所借的书

BorrowBookID	BorrowBookName	StudentID	BorrowBookPublish
1	马克思主义政治经济学	1	电子工业出版社
2	毛泽东思想概论	2	高等教育出版社
3	邓小平理论	3	人民邮电出版社
4	大学生思想道德修养	4	中国铁道出版社
5	C语言程序设计	5	高等教育出版社

Next

4.2.2 查询树

例子：多表连接操

Select Student.StudentName,Student.StudentAge,BorrowBook.BorrowBookName ,BorrowBook.BorrowBookPublish
FROM Student,BorrowBook
WHERE Student.StudentID = BorrowBook.StudentID

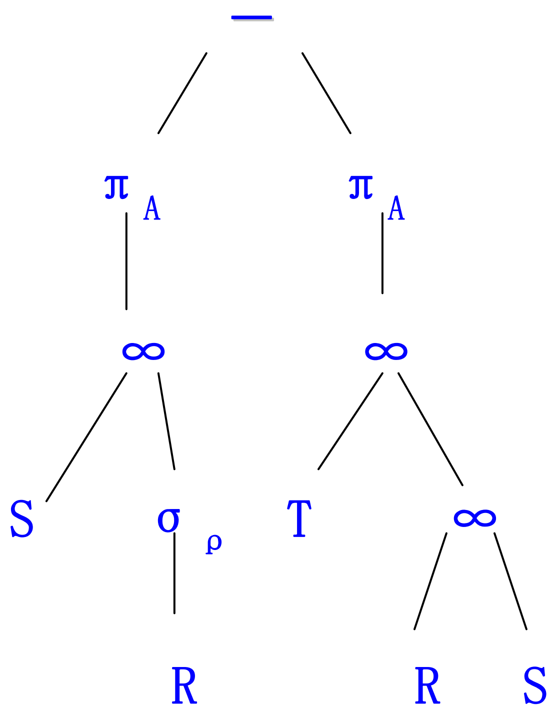
运行的结果如下：

StudentName	StudentAge	BorrowBookName	BorrowBookPublish
张三	25	马克思主义政治经济学	电子工业出版社
李四	26	毛泽东思想概论	高等教育出版社
王五	27	邓小平理论	人民邮电出版社
赵六	28	大学生思想道德修养	中国铁道出版社

4.2.2 查询树

用查询树表示更加复杂的查询表达式：

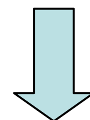
$$E=\pi_A(S\infty\sigma_\rho(R))-\pi_A(T\infty R\infty S)$$



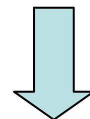
查询树可以理解为全局查询树，其叶节点是全局关系。

根据等价定义，可用三种方式描述查询：

SQL表达式（查询语句）



代数表达式



查询树

注意:查询树不同于分解树

4.2.2 查询树

$R_0(U_0,Q_0,S_0)(H)$

$R_1(U_0,Q_1,S_1)(M)$ $R_2(U_0,Q_2,S_2)(V)$ $R_3(U_0,Q_3,S_3)$

$R_{11}(U_{11},Q_{11},S_{11})$ $R_{12}(U_{12},Q_{12},S_{12})$ $R_{21}(U_{21},Q_2,S_{21})$ $R_{22}(U_{22},Q_2,S_{22})(V)$

$R_{221}(U_{221},Q_2,S_{221})$ $R_{222}(U_{222},Q_2,S_{222})$

图 分解树

4.2.3 等价变换规则

利用等价性定义，等价规则可以归纳为两类

（1）单个关系的代数操作的变换规则

$R \cup R \equiv R$	$R \cup \emptyset \equiv R$	$\emptyset - R \equiv \emptyset$	$R \propto \emptyset \equiv \emptyset$
$R \cap R \equiv R$	$R \cap \emptyset \equiv \emptyset$	$R \times \emptyset \equiv \emptyset$	$\emptyset \propto R \equiv \emptyset$
$R \infty R \equiv R$	$R \infty \emptyset \equiv \emptyset$	$\sigma_P(\emptyset) \equiv \emptyset$	
$R - R \equiv \emptyset$	$R - \emptyset \equiv R$	$\pi_A(\emptyset) \equiv \emptyset$	

其中，关系R与空关系进行操作（联接）表示了一种空操作，在查询转换过程中是可以消去的操作（某种程度的优化）

例4.2： $(R - R) \infty (R \cup R) \Rightarrow \emptyset \infty R \Rightarrow \emptyset$

4.2.3 等价变换规则

（2）多个关系模式的操作的变换规则

设有多个关系，其关系模式分别为**R**， S，T，在一定条件下有如下规则：

① 一元操作交换律： $U_1(U_2(R)) \equiv U_2(U_1(R))$

② 二元操作结合律： $(R)B((S)B(T)) \equiv ((R)B(S))B(T)$

③ 二元操作交换律： $(R)B(S) \equiv (S)B(R)$

④ 一元操作幂等律： $U(R) \equiv U_1U_2(R)$

⑤ 一元操作对二元操作的分配律： $U((R)B(S)) \Rightarrow (U(R))B(U(S))$

⑥ 一元操作的因式分解律： $(U(R))B(U(S)) \Rightarrow U((R)B(S))$

利用等价变换规则可以改变操作序实现查询优化

4.3 全局查询到逻辑查询的转换

讨论“查询转换”，是讨论分布式数据库查询处理的优化。即在转换过程中，利用等价变换规则，综合并充分地考虑分布查询的代价因素，使分布查询处理逐步实现优化。

4.3.1 全局查询到逻辑查询的转换步骤

4.3.2 等价转换准则

4.3.2.1 全局查询转换成查询树

4.3.2.2 生成优化的逻辑查询树

4.3.1 全局查询到逻辑查询的转换步骤

原则上可以按两步实现分布式查询的第一次转换（全局查询到逻辑查询），每步可遵守一些转换规则，以实现部分优化：

第一步：

- 将全局查询表达式（**SQL**语法和代数操作表达式）转换成全局查询内部结构形式（查询树）
- 在其转换过程中需要利用等价变换规则及其所归纳出来的两个转换准则（**C1**，**C2**）

第二步：

- 将全局查询树与模式分解树合并转换成部分优化的逻辑关系查询树，或称分解树的化简操作。其中包括：将全局查询树叶节点按分片模式定义的逻关系名，取代全局关系名（查询树与分解树合并），并分别运用变换规化简成部分优化的逻辑查询树。
- 其实现过程中，除了应用转换准则**C1**，**C2**以外，还有**C3**～**C6**准则。

4.3.2.1 全局查询转换成查询树

[例4.4]：对例**4.1**的查询树**(a)**，利用代数操作等价变换规则可有如图**4.4**所示。

(a)

$\pi_{S\#}$

$\sigma_{AREA='North'}$

$\infty_{D\#=D\#}$

SP Dept

(b)

$\pi_{S\#}$

$\infty_{D\#=D\#}$

SP $\sigma_{AREA='North'}$

Dept

(c)

$\pi_{S\#}$

$\infty_{D\#=D\#}$

$\pi_{S\#, D\#}$ $\pi_{D\#}$

SP $\sigma_{AREA='North'}$

Dept

图4.4 全局查询树转换范例

4.3.2.1 全局查询转换成查询树

分析：

图**4.4（a）**是下列代数表达式的查询树：

$$\pi_{S\#}\ \sigma_{AREA='North'}\ (Sp \underset{D\#=D\#}{\infty} Dept)$$

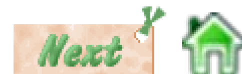
图4.4（b）是利用一元操作对二元操作的分配律规则：**U((R)B(S)) ⇒ (U(R))B(U(S))**

把一元操作向下移动，这时的代数操作表达式为：

$$\pi_{S\#}\ (Sp \underset{D\#=D\#}{\infty} \sigma_{AREA='North'}\ (Dept))$$

图**4.4（c）**是利用一元操作幂等律：**U(R) ≡ U1 U2 (R)** 对"操作数关系"分解为多个一元操作，以缩减"操作数关系"。即通过替换运算得：

$$\pi_{S\#}\ (\pi_{S\#,\ D\#}\ (Sp) \underset{D\#=D\#}{\infty} \pi_{S\#}\ \sigma_{AREA='North'}\ (Dept))$$



4.3.2.1 全局查询转换成查询树

结论：

查询树相当于对一个集中式数据库的查询，集中式数据库的逻辑优化技术可作用于其上。具体说是要：

①对全局查询树中的一元操作尽量下移到叶节点；

②如果查询树中有二元操作，则应尽量先缩减二元操作的操作数。由此，可得等价转换准则**C1**和**C2**。

准则C1：缩减二元操作数关系，利用一元操作对二元操作的分配律，将一元操作向下移动。

U((R)B(S)) ⇒ (U(R))B(U(S))

准则C2：用一元操作幂等律对操作数关系产生适当的一元操作或分解成多个一元操作，以缩减操作数关系。

$\mathbf{U(R) \equiv U_1 U_2\ (R)}$

4.3.2.2 生成优化的逻辑查询树

生成优化的逻辑查询树，就是把查询树与全局模式的分解树一起来考虑，需要用到限定关系等价变换性质。这一步的操作比较复杂，基本上分为以下几个子过程：

4.3.2.2.1 分解树的化简处理过程

4.3.2.2.2 分解树与查询树的合并过程

4.3.2.2.3 分布联接的化简过程

4.3.2.2.4 一个实例

4.3.2.2.1 分解树的化简处理过程

分解树表明了全局关系由哪些逻辑关系组成，按什么方式组合 。

要将全局查询转换成逻辑查询，需要对分解树进行处理。对于一个全局查询而言，并非构成全局关系的所有逻辑关系都将涉及到，往往只使用其中某一些，所以应根据查询树对分解树进行处理。我们称它为分解树的化简处理。

$R_0(U_0,Q_0,S_0)(H)$

$R_1(U_0,Q_1,S_1)(M)$ $R_2(U_0,Q_2,S_2)(V)$ $R_3(U_0,Q_3,S_3)$

$R_{11}(U_{11},Q_{11},S_{11})$ $R_{12}(U_{12},Q_{12},S_{12})$ $R_{21}(U_{21},Q_2,S_{21})$ $R_{22}(U_{22},Q_2,S_{22})(V)$

$R_{221}(U_{221},Q_2,S_{221})$ $R_{222}(U_{222},Q_2,S_{222})$

图 分解树结

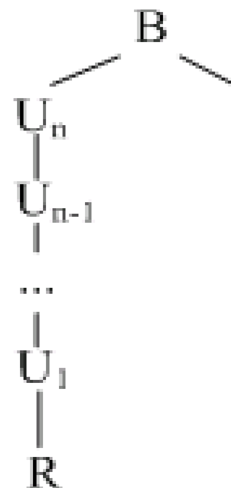
4.3.2.2.1 分解树的化简处理过程

当全局查询用查询树表示，经过**C1**、**C2** 准则处理后，查询要用到的逻辑关系的条件在查询树中就表现出来了。接着，可以根据这些条件消去一些与查询无关的逻辑关系，即去掉一些操作为空的子树。

假设在查询树中，存在一棵关于全局关系**R**（**U**，**True**，**T**）的一元操作 $U_n U_{n-1} \ldots U_1$ 的子查询树。令**F**为 U_l，…U_n中所有选择操作谓词的逻辑合取，即有 **F=** $\wedge_i P_i(U_i=\sigma_{pi})$ 。如果没有选择操作，则**F=True**，令**A**为U_1,…，U_n中所有投影操作中的属性和谓词P_j中所涉及的属性的并，即有：

$$A = (\cup_i A_i) \cup (\cup_j A(P_j)),\ U_j = \sigma_{pj}, U_i = \pi_{Ai};$$

令$Q_s=U_n,\ldots,U_1(R)$为关系R上的子查询，下面给出分解树化简的定义：



定义4.4： 一个关系R_i（U_i，Q_i，S_i）对于子查询Q_s是无用的，当 $F \wedge Q_i$=False，$U_i \cap A=Ø$；

否则是有用的。如果R_i是诱导分片所得的关系，当其主关系是无用的，它也是无用的。

4.3.2.2.1 分解树的化简处理过程

例4.5 ①设有关系R_0上的子查询Qs= $\pi_{A2}\sigma_{P2}\pi_{A1}\sigma_{p1}$ (R_0)

其查询子树如图4.7（a）所示，②R_0的分解树见图3.6，有 $F=P_1 \wedge P_2$，$A=A_1 \cup A_2 \cup A（P_1）\cup A（P_2）$。

假设有$F \wedge Q_1$=Flase，$A \cap U_{221}$=Ø,则③化简后的分解树如图4.7（b）所示。

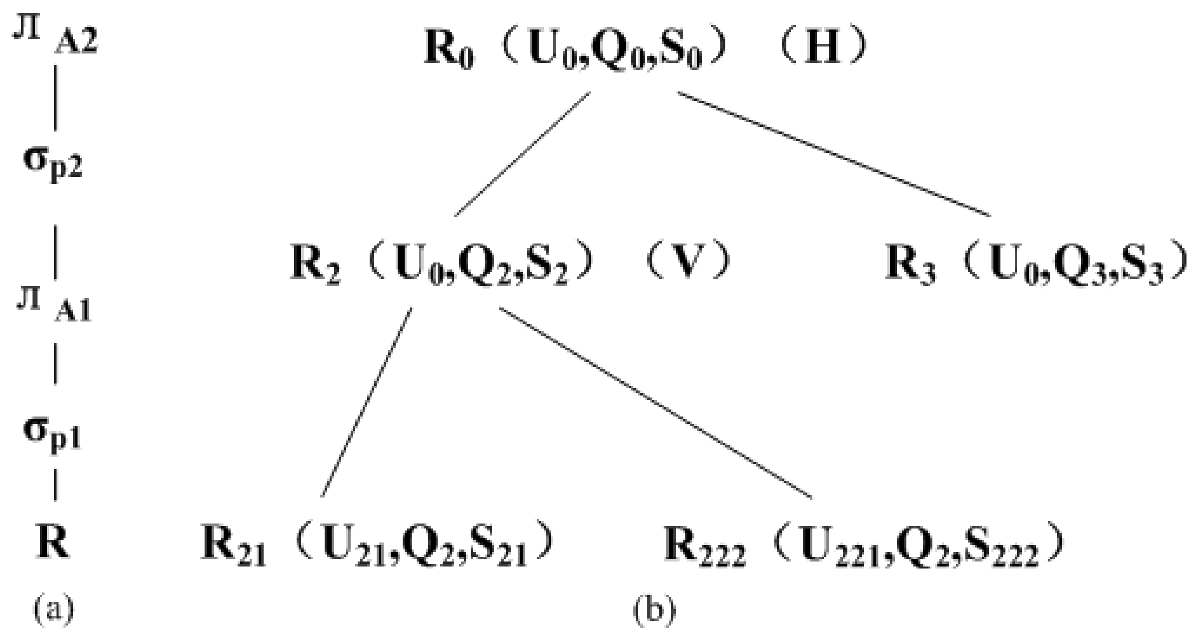
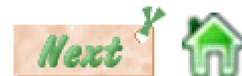


图4.7 分解树化简



4.3.2.2.1 分解树的化简处理过程

$R_0(U_0,Q_0,S_0)(H)$

$R_1(U_0,Q_1,S_1)(M)$ $R_2(U_0,Q_2,S_2)(V)$ $R_3(U_0,Q_3,S_3)$

$R_{11}(U_{11},Q_{11},S_{11})$ $R_{12}(U_{12},Q_{12},S_{12})$ $R_{21}(U_{21},Q_2,S_{21})$ $R_{22}(U_{22},Q_2,S_{22})(V)$

$R_{221}(U_{221},Q_2,S_{221})$ $R_{222}(U_{222},Q_2,S_{222})$

图3.6 独立分片操作后的分解树结构

4.3.2.2.1 分解树的化简处理过程

根据上述论述，可从中归结两个化简分解树时有用的准则**C3**，**C4**：

■准则 **C3**：在全局查询转换成逻辑查询的过程中，可以消去谓词合取具有矛盾的子树，即可消去选择操作结果为空的子查询树。

■准则**C4**：在全局关系转换成逻辑关系查询过程中，也可以消去联接操作结果为空的子树。

4.3.2.2.2 分解树与查询树的合并过程

在分解树简化处理后应该与查询树合并。

这是两种性质不同的树，但由于分解树是由全局关系经过分片操作形成的，一组代数操作。查询树是对全局关系的查询操作，也是一组代数操作。所我们可以用以下算法实现转化。

算法4.2：简化的分解树转化为逻辑查询树。

输入：已经化简后的分解树。

输出：从全局查询转换为逻辑查询树。

方法：从根节点开始：

(1) 如果节点上操作**Oj=H**或**DH**，则将其转换为**U**（一元）操作节点；

(2) 如果节点上的操作**Oj=V**，则将其转换为联接操作节点，联接属性是**k**；

(3) 如果节点操作**Oj=AO**，则不必转换；

(4) 直至将所有节点处理完毕，最后输出（获得）对逻辑片段的查询树。

(5) 当然，当得到了逻辑片段(关系)的查询树后，还应按**C1**、**C2**准则反复变换，使得一元操作下移，二元操作的操作数尽量缩减。

4.3.2.2.3 分布联接的化简过程

所谓分布联接是指具有两个以上（含两个）全局关系的联接（特别是它们不在同一场地上）。

例如：在查询树中，如果有两个全局关系R，S联接时，对R，S的所有元组都应进行比较；当这两个全局关系的逻辑关系不在同一场地上，就须经通讯形成分布联接。图4.8中，节点表示全局关系的逻辑关系（分片后），边表示两节点间联接不为空。

图4.8（a）是R，S的全联接图，即R的所有逻辑关系（R_1，…，R_n）与S的所有逻辑关系（s_1，…，S_n）进行完全分布联接。对于DDB来讲，这种联接的代价是极大的。所以，在设计DDB时，对于有两个联接操作的关系（常常体现在实体间的关联性质）应尽量使其划分合理。

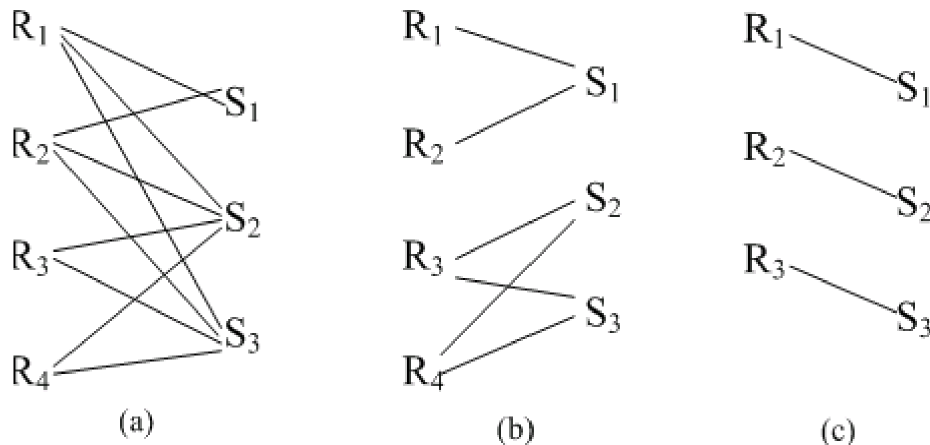
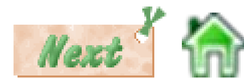


图4.8 分布联接图



4.3.2.2.3 分布联接的化简过程

对于完全联接的化简法有两种：

- 一种是部分分布联接图［如图**4.8**（**b**）］，其中部分节点间没有联通，使完全联接图形成两个或两个以上子图。
- 另一种是简化为简单分布联接图［如图**4.8**（**C**）］，每对节点间只有一条边。

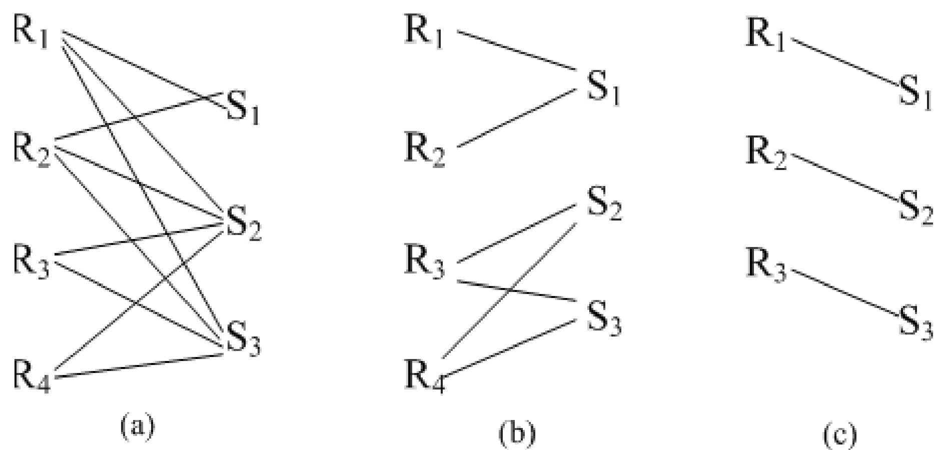
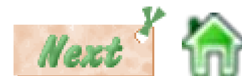


图4.8 分布联接图



4.3.2.2.3 分布联接的化简过程

DDB中分片操作支持的诱导操作实际上是这种简单分布联接，R与S的逻辑关系只存在一对一的联接，这就可以先做局部联接，再完成分布联接，其通讯开销一定会降低。所谓先做局部联接，就是先在逻辑关系之间完成联接，然后再合并。

例4.6　设有图4.9（a）所示的查询树，S_1，S_2是按R_1，R_2诱导分片操作所得到的逻辑关系，该查询树可以依等价变换规则转换成图4.9（b）所示。图4.9（c）表示简单分布联接的查询树。

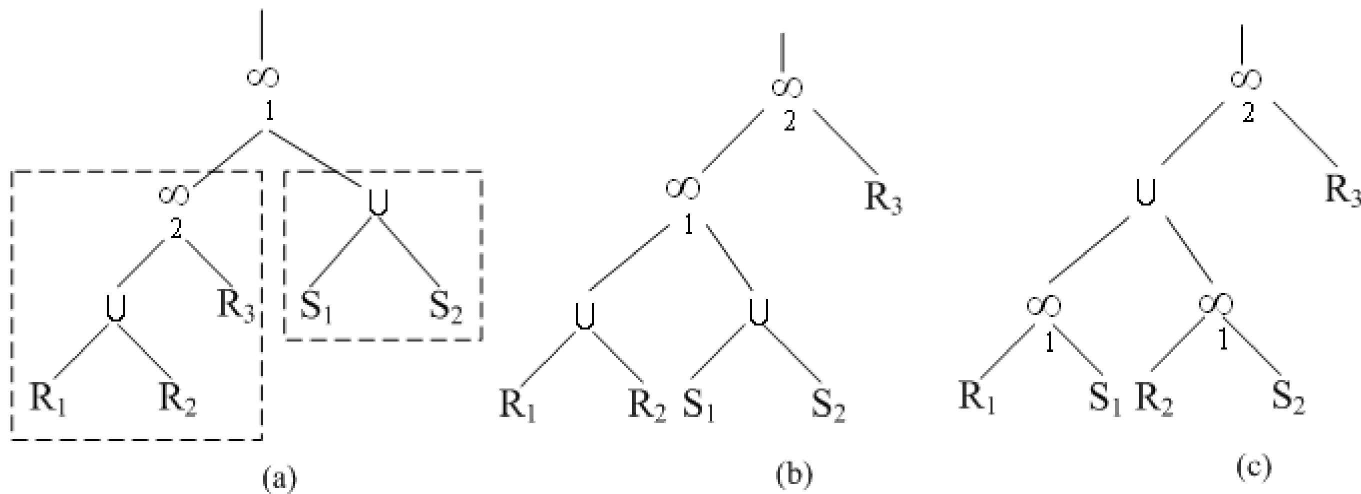
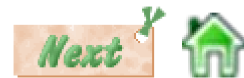


图4.9 简单分布联接



内容回顾（3.2.2.3.4诱导分片）

Supplier

SNO	SNAME	SCITY
S1	Smith	London
S2	Jones	Paris
S3	Blake	Paris

Supplier1(T1)

SNO	SNAME	SCITY
S1	Smith	London

Supplier2(T2)

SNO	SNAME	SCITY
S2	Jones	Paris
S3	Blake	Paris

Supply

SNO	PNO	QTY
S1	P1	320
S1	P2	200
S1	P3	450
S2	P1	300
S2	P2	250
S3	P2	100

Supply1(R1)

SNO	PNO	QTY
S1	P1	320
S1	P2	200
S1	P3	450

Supply2(R2)

SNO	PNO	QTY
S2	P1	300
S2	P2	250
S3	P2	100

图 诱导分片范例

4.3.2.2.3 分布联接的化简过程

从例**4.6**我们可以看到：对于图**4.9**（**c**）的查询树表示了先做联接操作再做并操作，有利于进一步优化查询树，其中使用了一些等价变化规则。所以可归纳出分布联接的转换准则**C5**。

准则C5：在全局查询中具有分布联接时，可将联接下属的并操作上推。

附：

② 二元操作结合律： **(R)B((S)B(T)) ≡ ((R)B(S))B(T)**

③ 二元操作交换律： **(R)B(S) ≡(S)B(R)**

4.3.2.2.4 一个实例

例子：至此，我们可以将例4.4的全局查询转换再根据上述准则进行优化。假设全局关系**Dept**按部门号水平分片，其谓词为：

Q1：D＃＝1～10

Q2：D＃＝11～20

Q3：D＃＝21～30

且**D＃=1～10**在“**North**”地区。同时有约定；**North**地区的零件由**London**供应者供应。图**4.10**是利用上列准则对图 **4.4**的进一步转换。

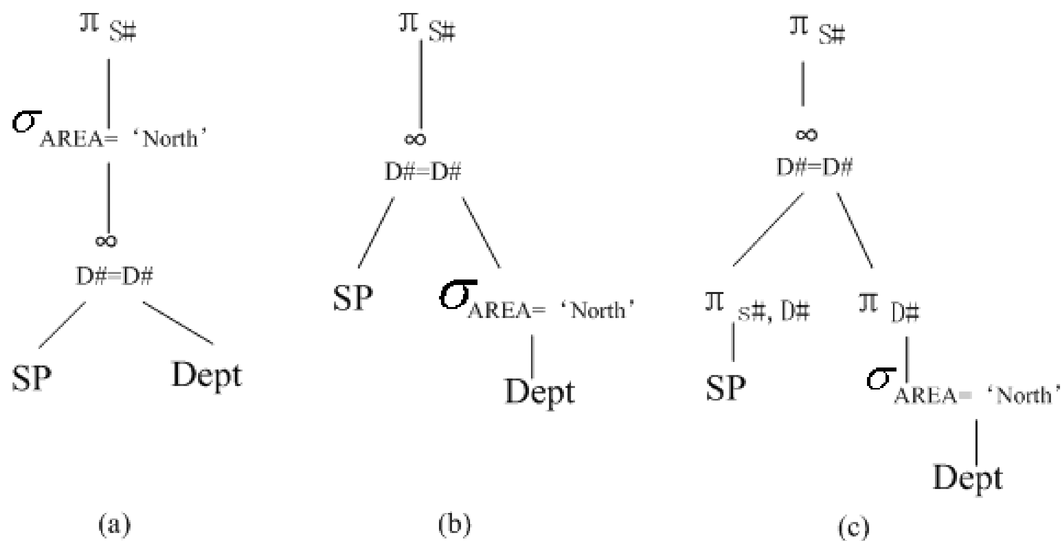


图4.4 全局查询树转换范例

4.3.2.2.4 一个实例

Πs#
∞
D#=D#
Πs#，D#
ΠD#
∪
σarea='North'
SP1 SP2
∪
Dept1 Dept2 Dept3
(a)

Πs#
∞
D#=D#
(1) ∪
∪ (2)
Πs#，D# Πs#，D# ΠD# ΠD# ΠD#
SP1 SP2 σarea='North' σarea='North' σarea='North'
Dept1 Dept2 Dept3
(b)

Πs#
∞
D#=D#
Πs#，D#
ΠD#
SP1
σarea='North'
Dept1
(c)

准则C1：缩减二元操作数关系，利用一元操作对二元操作的分配律，将一元操作向下移动。

准则C2：用一元操作幂等律对操作数关系产生适当的一元操作或分解成多个一元操作，以缩减操作数关系。

准则 C3：在全局查询转换成逻辑查询的过程中，可以消去谓词合取具有矛盾的子树，即可消去选择操作结果为空的子查询树。

准则C4：在全局关系转换成逻辑关系查询过程中，也可以消去联接操作结果为空的子树。

准则C5：在全局查询中具有分布联接时，可将联接下属的并操作上推。

图4.10是例4.l的全局查询到逻辑查询的最后查询树，其中经过了从C1～C5准则。

4.4 逻辑查询到物理查询的转换

一、什么是物理查询转换？

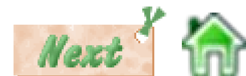
•所谓从逻辑查询到物理查询转换，是指将逻辑查询转换得到的简化了的查树，转换成具体的每个场地上的局部操作命令和数据传输命令的过程。

•全局查询须两次转换后才形成一个具体的分布执行计划（**DEP**），然后才交付给各个局部场地去执行。

二、物理转换的基本内容和策略

物理查询转换的过程，将涉及到物理副本和查询的处理场地，即执行环境。特别对于二元操作数不在同一场地时或者有多个副本可选择的情况时，其“执行环境”的概念更为重要。所以，物理转换时一般注意以下因素：

●操作副本选择
●操作执行次序的选择
●操作方法的选择
●通讯代价
●评估数据量



4.4 逻辑查询到物理查询的转换

■操作副本的选择是选定逻辑关系相对应的物理关系有多个副本时的具体化。原则上，对不同查询有不同的具体选择。各个物理关系的副本其使用情况、径代价和使用要求不完全一样，若按随机选定显然不合理，应该遵守一定的定，选择一个理想的（合理的）副本。

副本选择的一般原则：

- 本场地物理关系优先。如果查询始发场地上有逻辑关系的一个相应的物理关系，就应尽量的选择该物理关系
- 同场地上有二元操作，则应尽量选择同一场地上的相应物理关系完成二元操作，以减少数据传送
- 在查询等候中数据最小的物理关系应被优先选中
- 代价最小的应优先选中

关于操作方法的选择，更多地取决于对局部数据库的存取方式

- 在物理转换时应尽量注意到对同一次数据库存取中的一些代数操作是否能组合在一起完成
- 尽量避免多次内/外存调用，这与局部层优化有很大关系

4.5 基于半联接的查询优化

4.5.1 联接操作重要性

4.5.2 联接操作

4.5.3 半联接操作原理和不对称性

4.5.4 半联接操作的缩减作用

4.5.5 半联接程序的作用

4.5.6 半联接程序的具体过程

4.5.1 联接操作重要性

- 关系数据库由许多关系组成，关系与关系之间的联系主要通过联接操作表现出来，因而在二元操作中，联接操作远比其它操作用得多。
- 讨论联接，其实包括了“选择——投影——联接”的综合问题，即二元操作和一元操作的综合优化问题。
- 分布式查询处理的联接操作，更是影响分布式查询效率的最关键因素。
- 在DDB中，联接操作的大量数据会引起场地间的传输，它直接影响整个系统性能。
- 当前对联接操作的优化有两种趋向：
  - ✓一种是采用半联接技术来减少联接操作的操作数，以降低通讯费用；
  - ✓另一种是直接进行联接操作的代价计算

4.5.2 联接操作

联接操作是从两个关系的笛卡尔积中选取属性间满足一定条件的元组。记作：

$$R \underset{A\theta B}{\bowtie} S = \{ \widehat{t_r t_s} \mid t_r \in R \land t_s \in S \land t_r[A] \,\theta\, t_s[B] \}$$

其中**A**和**B**分别为**R**和**S**上可比的属性组。

自然联接（**Natural join**）是一种特殊的等值联接，它要求两个关系中进行比较的分量必须是相同的属性组，并且要在结果中把重复的属性去掉。即若**R**和**S**具有相同的属性组**B**，则自然连接可记作：（实例）

$$R \bowtie S = \{ \widehat{t_r t_s} \mid t_r \in R \land t_s \in S \land t_r[B] = t_s[B] \}$$

等值连接（**equi-join**），θ为"＝"的连接运算称为等值连接。它是从关系**R**与**S**的笛卡尔积中选取**A**、**B**属性值相等的那些元组。即等值连为：（实例）

$$R \underset{A=B}{\bowtie} S = \{ \widehat{t_r t_s} \mid t_r \in R \land t_s \in S \land t_r[A] = t_s[B] \}$$

（回顾）自然联接

自然联接的结果是在R和S中的在它们的公共属性名字上相等的所有元组的组合。例如下面是表格"雇员"和"部门"和它们的自然联接:

雇员

Name	EmpId	DeptName
Harry	3415	财务
Sally	2241	销售
George	3401	财务
Harriet	2202	销售

部门

DeptName	Manager
财务	George
销售	Harriet
生产	Charles

雇员 ⋈ 部门

Name	EmpId	DeptName	Manager
Harry	3415	财务	George
Sally	2241	销售	Harriet
George	3401	财务	George
Harriet	2202	销售	Harriet

图 自然联接实例

（回顾）等值联接

θ-联接和等值联接

如果我们要组合来自两个关系的元组，而组合条件不是简单的共享属性上的相等，则有一种更一般形式的连接算子才方便，这就是θ-联接(或theta-联接)。θ 是在集合{<, ≤, =, >, ≥}中的二元关系。θ的结果由在 R 和 S 中满足关系θ 的元素的所有组合构成。只有 S 和 R 的表头是不相交的，即不包含公共属性的情况下，θ-连接的结果才是有定义的。

实例：考虑分别列出车模和船模的价格的表“车”和“船”。假设一个顾客要购买一个车模和一个船模，但不想为船花费比车更多的钱。在关系上的θ-联接 $CarPrice \geq BoatPrice$ 生成所有可能选项的一个表。

车

CarModel	CarPrice
CarA	20'000
CarB	30'000
CarC	50'000

船

BoatModel	BoatPrice
Boat1	10'000
Boat2	40'000
Boat3	60'000

车 $\bowtie_{CarPrice \geq BoatPrice}$ 船

CarModel	CarPrice	BoatModel	BoatPrice
CarA	20'000	Boat1	10'000
CarB	30'000	Boat1	10'000
CarC	50'000	Boat1	10'000
CarC	50'000	Boat2	40'000

图 θ-联接实例

4.5.3 半联接操作原理和不对称性

半联接操作是关系代数操作中联接（JOIN）操作的一种缩减，关系R和S半联接记为R∝S。<u>其结果关系是R和S的自然联接（Natural JOIN）后，R的属性上的投影</u>，可用下述表达式表示：

$R \propto S = \pi_R (R \infty S)$

等价方法：将S中与R有相同属性名的属性集投影出来，然后与R完成自然联接，其等价公式为：

$R \propto S = R \infty (\pi_B S)$

具不对称操作性：$R \propto S \neq S \propto R$。

半联接操作是一种导出操作：

一个关系的分片是根据另一个与其有关联性质的关系的属性来划分，即诱导分片。而诱导分片就是用“半联接”概念实现的，即诱导分片是两个相关关系的半联接产生的。或者具体地说，是两个相关关系实现自然联接后，在主关系的属性上的投影。

一个半联接操作的实例

（回顾）半联接

半联接的结果只是在 *S* 中有在公共属性名字上相等的元组所有的 *R* 中的元组。

实例：“雇员”和“部门”和它们的半联接的表：

雇员

Name	EmpId	DeptName
Harry	3415	财务
Sally	2241	销售
George	3401	财务
Harriet	2202	生产

部门

DeptName	Manager
销售	Harriet
生产	Charles

雇员 ⋉ 部门

Name	EmpId	DeptName
Sally	2241	销售
Harriet	2202	生产

图 半联接实例

4.5.4 半联接操作的缩减作用

例4.17 有R（A,B）和 S（B,C），根据半联接计算公式有：$R \underset{B}{\propto} S = R \infty (\pi_B S)$ 和 $S \underset{B}{\propto} R = S \infty (\pi_B R)$ 。如果有图 4.21（a）的实例，则 R $\underset{B}{\propto}$ S的结果如 4.21（b）所示，S $\underset{B}{\propto}$ R的结果如图4.21（C）所示。

R（A,B）

A	B
a1	b1
a2	b1
a2	b3
a2	b4
a3	b3

S（B,C）

B	C
b1	c1
b2	c2
b5	c3
b5	c4
b5	c5
b6	c6
b7	c7
b8	c8

(a) 关系R（A,B）和S（B,C）

R′

A	B
a1	b1
a2	b1

(b) $R \underset{B}{\propto} S = R\infty(\pi_B S)$

S′

B	C
b1	c1

(c) $S \underset{B}{\propto} R = S\infty(\pi_B R)$

从图4.21（b）、（c）可得出结论：半联接操作对操作数R或S有缩减作用，且由于其不对称性则各自缩减的程度不一样。半联接操作的缩减性与在联接操作前先对操作数关系做选择和投影有相似的效果。特别在分布式环境中，可用半联接操作减少网上数据的传送量。

图4.12 半联接操作的不对称性与缩减作用

4.5.5 半联接程序的作用

半联接程序是用半联接技术实现联接操作的程序，即用一组具有半联接与联接的操作，映射出具有与等值联接相同结果的过程。

$$R \underset{A=B}{\infty} S \Leftrightarrow (R \underset{A=B}{\propto} \pi_B S) \underset{A=B}{\infty} S \qquad \text{(4-11a)}$$

$$R \underset{A=B}{\infty} S \Leftrightarrow (S \underset{B=A}{\propto} \pi_A R) \underset{B=A}{\infty} R \qquad \text{(4-11b)}$$

（**4－11a**）、（**4－11b**）式说明半联接程序与两个关系的直接等值联接等价。

同样，在分布式数据库中，当**R**，**S**两个关系不在相同场地上时，用（**4－11a**）公式或（**4－11b**）公式处理，可以减少联接操作的数据传送量，并且半联接程序的技术可以缩减它的操作数。

4.5.6 半联接程序的具体过程

以式（4-11a）为例，且假定 R和S不在同一场地：

① 在s场地对S关系做投影操作，使S关系缩减为S′：

$$\pi_B S \Rightarrow S'$$

② 将S′ 送往r场地；

③ 在r场地上完成R与S′ 的半联接操作，使R关系缩减为R′：

$$R \underset{A=B}{\propto} S' \Rightarrow R'$$

④ 将R'关系送回S场地；

⑤ 在S场地完成R'与S的联接操作。

$$T = R' \infty S$$

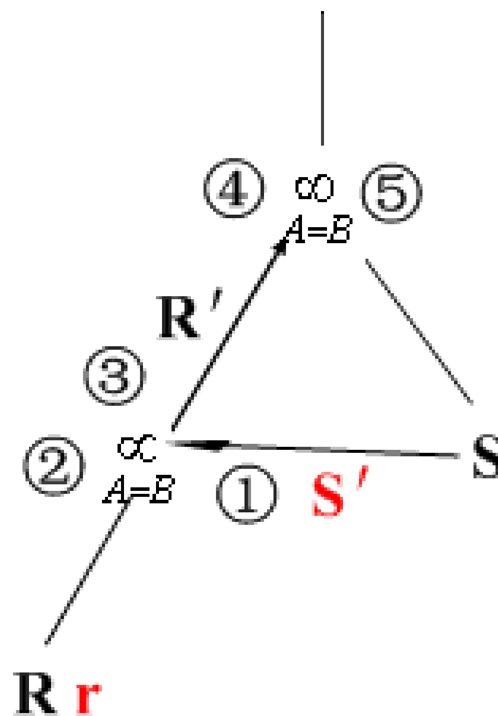


图4.22 半联接程序操作

图4.22的核心技术思想是只将联接操作有关的操作分量在网上进行传输。R与S关系在A=B条件下联接，R、S关系只有公共属性值相等的那些元组才有意义。

4.6 基于枚举法的查询优化

- 4.6.1 概述
- 4.6.2 嵌套循环联接算法
- 4.6.3 基于排序的联接算法
- 4.6.4 散列连接算法

4.6.1 概述

- 半联接优化方法能够减少查询执行的通信代价，但是同时会导致通信次数的增加和局部执行代价的增加。
- 当系统环境为高速局域网时，查询执行代价主要考虑的是局部处理代价，半联接优化方法则不再适用。
- 在这种情况下，分布式数据库系统通常使用基于直接联接技术的枚举法优化技术。
- 所谓枚举法优化，就是枚举联接操作所有可行的直接联接算法，通过对每种方法的查询执行代价估计，从中选择一种代价最小的算法作为联接操作的执行算法。
- 直接联接算法广泛应用于集中式数据库系统中，包括：嵌套循环连接算法、归并排序连接算法、散列连接算法和基于索引的连接算法。

4.6.2 嵌套循环联接算法

- 4.6.2.1 基于元组的嵌套循环联接
- 4.6.2.2 基于块的嵌套循环联接
- 4.6.2.3 嵌套循环联接方法的代价估计

4.6.2.1 基于元组的嵌套循环联接

- 嵌套循环连接算法是一种最简单的联接算法，其原理是对联接操作的两个关系对象中的一个仅读取其元组一次，而对另一个关系对象中元组将重复读取。
- 嵌套循环联接算法的特点是可以用于任何大小的关系间的连接操作，不必受连接操作所分配的内存空间大小的限制。
- 对于嵌套循环连接算法，可根据每次操作的对象大小分为基于元组的嵌套循环连接和基于块的嵌套循环连接。
- 假设有关系R(A,B)和关系S(B,C)，分别有Card(R)=n和Card(S)=m，现在要执行两个关系在属性B上的连接操作，如图4.13所示。

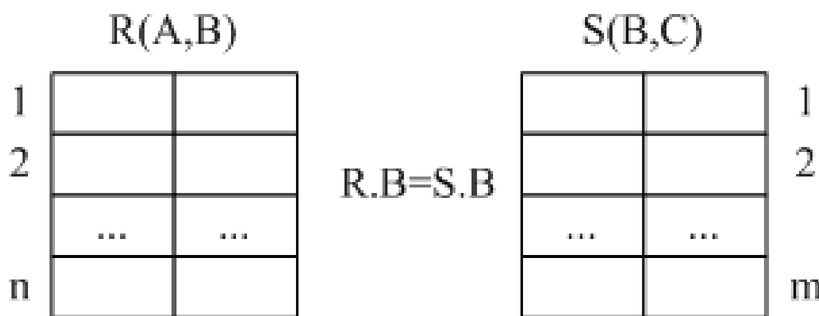
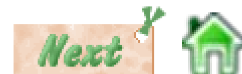


图4.13 两个联接关系

4.6.2.1 基于元组的嵌套循环联接

基于元组的嵌套循环连接

```
Result = {};/*初始化结果集合*/
For each tuple s in S
    For each tuple r in R
        If r.B=s.B then /*元组r和元组s满足连接条件*/
            Join r and  s as tuple t;
            Output t into Result;/*输出连接结果元组*/
        Endif
    Endfor
Endfor
Return Result
```



4.6.2.1 基于元组的嵌套循环联接

- 上面基于元组的嵌套循环连接算法中，对于循环外层的关系，通常称为外关系，而对循环内层的关系称为内关系。
- 在执行嵌套循环连接时，仅对外关系进行1次读取操作，而对内关系则需要进行反复读取操作。
- 如果不进行优化的话，这种基于元组的执行代价很大，以磁盘IO计算最多可能达到Card(R)*Card(S)。
- 因此，通常对这种算法进行修改，以减少嵌套循环连接的磁盘IO代价。一种方法是使用连接属性上的索引，以减少参与连接元组的数量；另一种方法是通过尽可能多地使用内存以减少磁盘IO数目。

4.6.2.2 基于块的嵌套循环联接

- 基于块的嵌套循环连接方法是通过尽可能多地使用内存，减少读取元组所需的I/O次数。其中，对连接操作的两个关系的访问均按块进行组织，同时使用尽可能多的内存来存储嵌套循环中的外关系的块。
- 与基于元组的方法类似，将连接操作中的一个对象作为外关系，每次读取部分元组到内存中，整个关系只读取一次，而另一个对象作为内关系，反复读取到内存中执行连接。
- 对于每个逻辑操作符，数据库系统都会分配一个有限的内存缓冲区。假设为连接操作分配的内存缓冲区大小为M个块，同时有Block(R)≥Block(S) ≥M，即连接的两个关系都不能完全读取到内存中。
- 为此，首先选择较小的关系作为外关系，这里选择关系S。将1到M-1块分配给关系S，而第M块分配给关系R。将外关系S按照M-1个块的大小分为多个子表，并将这些子表依次读取到内存缓冲区中，关系R的每个块会被重复地读入内存和关系S的子表进行连接。
- 对于内存缓冲区中元组的连接操作，先在M-1个块的外关系S元组的连接属性上构建查找结构，再从内关系R在内存中的块中读取元组，通过查找结构与S中的元组连接。

4.6.2.2 基于块的嵌套循环联接

Result={};/*初始化结果集合*/

Buffer=M;/*内存缓冲区*/

For each M-1 in Block(S)/*每次从外关系S中读取M-1个块到内存缓冲区中*/

　　Read M-1 of Block(S) into Buffer;

　　For each block in Block(R) /*每次从内关系R中读取1个块到内存缓冲区*/

　　　　Join M-1 of Block(S) and 1 of Block(R) in Buffer;/*在内存中对块中元组执行连接*/

　　　　Output t into Result;

　　Endfor

Endfor

Return Result;

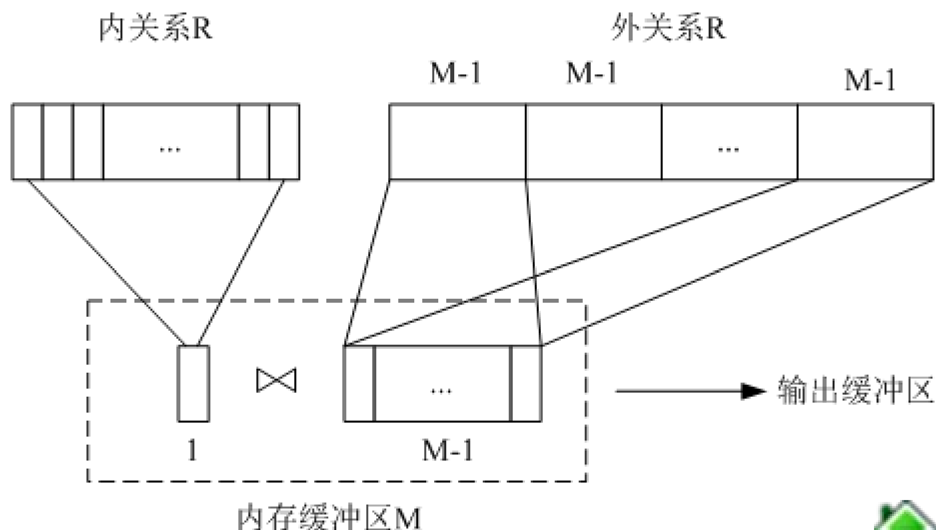


图4.14 基于块的嵌套循环连接算法示意图

4.6.2.3 嵌套循环联接方法的代价估计

- 对于两个关系R和S，如果使用基于元组的嵌套循环连接方法，则需要对每个元组的读取产生1次磁盘IO。因此，假设两个关系的元组数量分别为Card(R)和Card(S)，则基于元组的嵌套循环连接方法的执行代价为Card(R)*Card(S)，即两个关系大小的乘积。
- 如果使用基于块的嵌套循环连接方法，假设两个连接关系R和S占用的块分别为Block(R)和Block(S)，M为内存缓冲区大小。在嵌套过程中使用S作为外关系，每次迭代时首先读取M-1块S的内容到内存缓冲区中，再每次读取R的Block(R)中的1个块的内容到内存中与M-1块地S内容进行连接。因此，连接的代价可以用以下公式计算：
- C_{join}=(Block(S)/(M-1))*(M-1 +Block(R))
- 公式可以进一步简化为：
- C_{join}=Block(S) +(Block(S)/(M-1))*Block(R)
- 从上面公式可以看出，在选择较小的关系作为连接的外关系时，可以获得最小的执行代价，因此，通常选择较小的关系作为外关系。

4.6.3 基于排序的联接算法

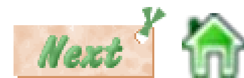
- 4.6.3.1 概述
- 4.6.3.2 归并排序算法
- 4.6.3.3 简单的基于排序的连接算法
- 4.6.3.4 归并排序连接算法

4.6.3.1 概述

- 基于排序的连接算法，首先将两个关系按照连接属性进行排序，然后按照连接属性的顺序扫描两个关系，同时，对两个关系中的元组执行连接操作。
- 由于数据库中关系的大小往往大于连接操作可用内存缓冲区的大小，因此，对关系的排序通常采用外存排序算法，即归并排序算法。
- 可以将基于排序的连接算法的执行过程与归并排序算法结合的算法，可以节省更多的磁盘IO，通常称为归并排序连接算法。

4.6.3.2 归并排序算法

- 简单的归并排序算法的执行可以分为两个阶段：
- **第一阶段**：对关系进行分段排序，即首先将需要排序的关系R划分为大小为M个块的子表，其中，M是可用于排序的内存空间的个数，以块为单位，再将每个子表放入内存中采用快速排序等主存排序算法执行排序操作，这样可以获得一组内部已排序的子表。
- **第二阶段**：对关系的子表执行归并操作，即按照顺序从每个排序的子表中读取一个块的内容放入内存，在内存中统一对这些块中的记录执行归并操作，每次选择最大（最小）的记录放入输出缓冲区中，同时删除子表中相应的记录。当子表在内存中的块被取空时，从子表中顺序读取一个新的块放入到内存中继续执行归并操作。



4.6.3.2 归并排序算法

- 归并排序的过程如图4.15所示，其中同时对多个子表执行归并操作，因此，也称为两阶段多路归并排序。需要说明的是，第二阶段的归并操作执行的条件是关系的子表数量小于排序操作可用的内存的块数M，这样才能保证同时对所有子表进行归并操作。
- 因此，两阶段归并执行的条件是关系的大小Block(R)≤M^2。如果关系的大小大于M^2，则需要嵌套执行归并排序算法，使用三阶段或更多次的归并操作。

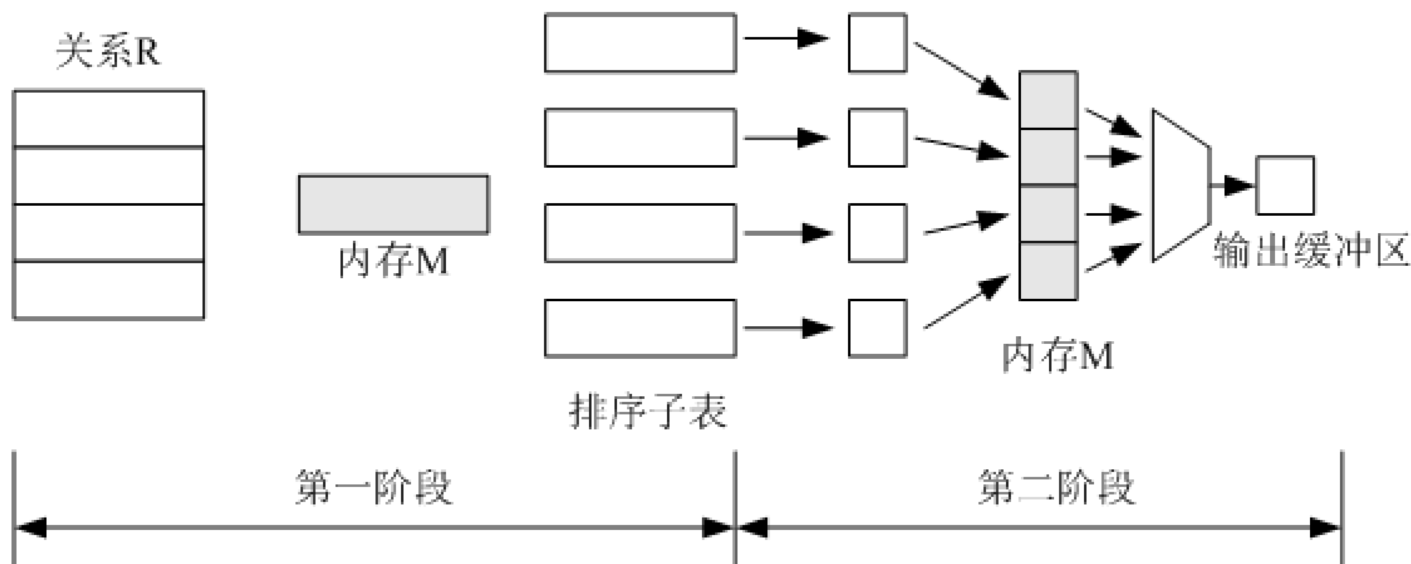


图4.15 简单的归并排序算法的执行

4.6.3.3 简单的基于排序的连接算法

- 基于排序的连接算法，主要是对已经按照连接属性排序的两个关系，按照顺序读取关系中的块到内存中执行连接操作。

代价计算

- 假设在排序阶段使用的是两阶段多路归并排序，关系的大小满足条件 $Block(R) \leqslant M^2$， $Block(S) \leqslant M^2$。这样，算法在排序阶段的执行代价包括对关系的子表执行排序所需的一次读（读子表数据）和一次写（子表排序结果写入磁盘）的代价2(Block(R)+Block(S))，以及多路归并时的读写代价2(Block(R)+Block(S))，而在归并连接阶段还需要对关系执行一次读操作。因此，简单的基于排序的连接算法的执行代价为：

  $C_{join}=5(Block(R)+Block(S))$。

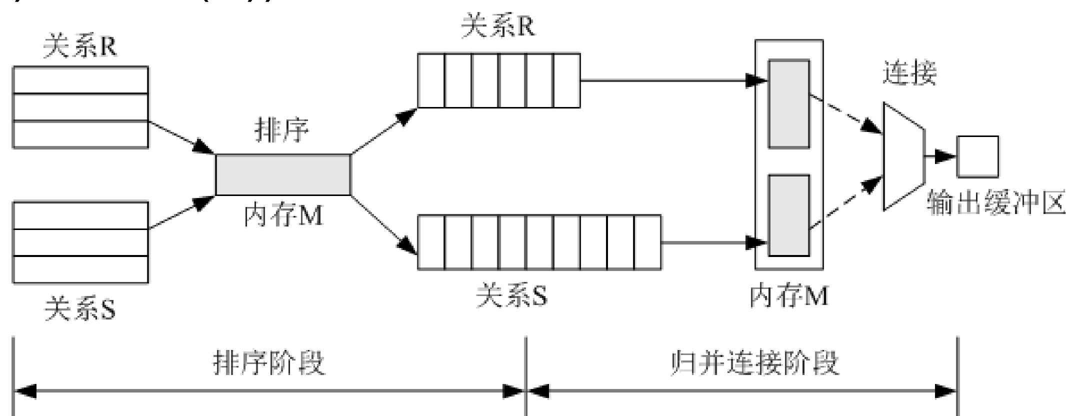


图4.16 简单的排序连接算法

4.6.3.4 归并排序连接算法

- 在简单的基于排序的连接算法中，归并连接阶段仅仅使用了内存缓冲区的两个块的空间，还有大量的空闲内存没有使用。因此，一种更加有效的归并排序连接算法被提出，其思想是将排序的第二阶段与归并连接阶段合并，即直接使用两个关系的排序子表执行归并连接操作，这样可以节省一次对关系的读写操作。
- 假设可用内存缓存区为M个块，算法首先对两个关系划分成大小为M个块的子表并排序，再从每个子表中顺序读取一个块调入内存缓冲区执行连接操作。这里要求两个关系的子表总数不超过M个。

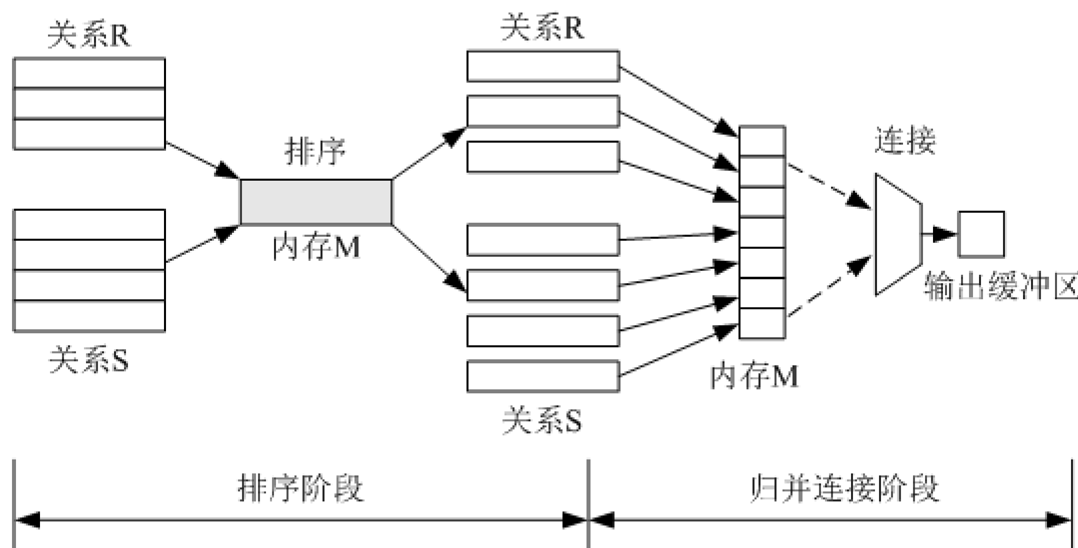


图4.17 归并排序连接算法示意图

4.6.3.4 归并排序连接算法

- 归并排序连接算法在排序阶段的代价包括对子表的一个读写操作2(Block(R)+Block(S))，而在归并连接阶段仅需一次代价为Block(R)+Block(S)的读操作，因此，执行的总代价为：

  C_{join}=3(Block(R)+Block(S))
- 这里注意，归并排序连接算法要求两个关系的子表数量，必须小于内存缓冲区的块数M，这样才能够保证归并阶段有足够的内存存放每个子表的一部分以执行连接。
- 因此，执行归并排序连接算法需要关系的大小满足：

  Block(R)+Block(S) ≤ M^2

4.6.4 散列连接算法

- 散列连接算法，也称为哈希连接算法，基本的执行过程同样分为两个阶段。
- 首先，使用同一个散列函数，对进行连接的两个关系R和S中的元组的连接属性值进行散列，在连接属性上具有相同键值的元组会出现在相同散列数值的桶中，然后，对两个关系中散列数值对应的桶中的元组执行连接操作。
- 假设可用的内存缓冲区为M块，散列时使用M-1个块作为桶的缓冲区（最多允许散列到M-1个桶），剩余的1个块作为扫描输入关系的缓冲区。

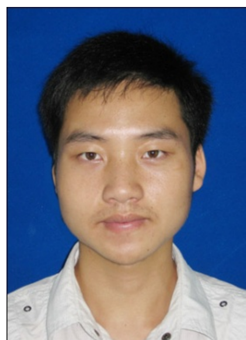
4.6.4 散列连接算法

- 在算法的第一个阶段中，使用内存将关系R和S散列到M-1个桶中，分别得到写入文件中的$R_1,\ldots,R_{m-1}$和$S_1,\ldots,S_{m-1}$，这个过程需要对两个关系执行一次读写操作，代价为2(Block(R)+Block(S))。
- 在第二个阶段中，每次选取两个关系中具有相同散列值的桶R_i和S_i放到内存中执行连接操作。假设S为较小的关系，由于在对桶连接时必须有一个桶能够全部装入M-1个内存缓冲区块中，才能够保证在执行桶连接时，保证仅执行一次读取操作。因此，关系S的大小需要满足：
  $Block(S) \leqslant (M-1)^2$
- 若连接的两个关系能够满足一次连接操作的条件，则散列连接算法的执行代价为：
  $C_{join}=3(Block(R)+Block(S))$

附件：主讲教师和助教

主讲教师：林子雨

单位：厦门大学计算机科学系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://www.cs.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn



助教：赖明星

单位：厦门大学计算机科学系数据库实验室2011级硕士研究生
E-mail: mingxinglai@gmail.com

Thank You!