第19讲 数据库查询实现算法-I

（一趟扫描算法）

本讲学习什么?

基本内容

1. 数据库查询实现算法概述
2. 以连接操作为例看逻辑实现算法与物理实现算法
3. 利用迭代器构造查询实现算法
4. 几个关系操作的一趟扫描算法
5. 基于索引的查询实现算法?

重点与难点

- 理解数据库查询实现的基本思想--逻辑算法和物理算法
- 理解查询实现算法与内外存环境的关系--如何利用内存
- 从物理存储上理解关系运算：一趟扫描算法
- 掌握关系运算的几个一趟扫描算法及其应用条件与算法复杂性

第19讲 数据库查询实现算法-I

19.1 数据库查询实现算法概述?

19.2 连接操作的实现算法--由逻辑层面到物理层面?

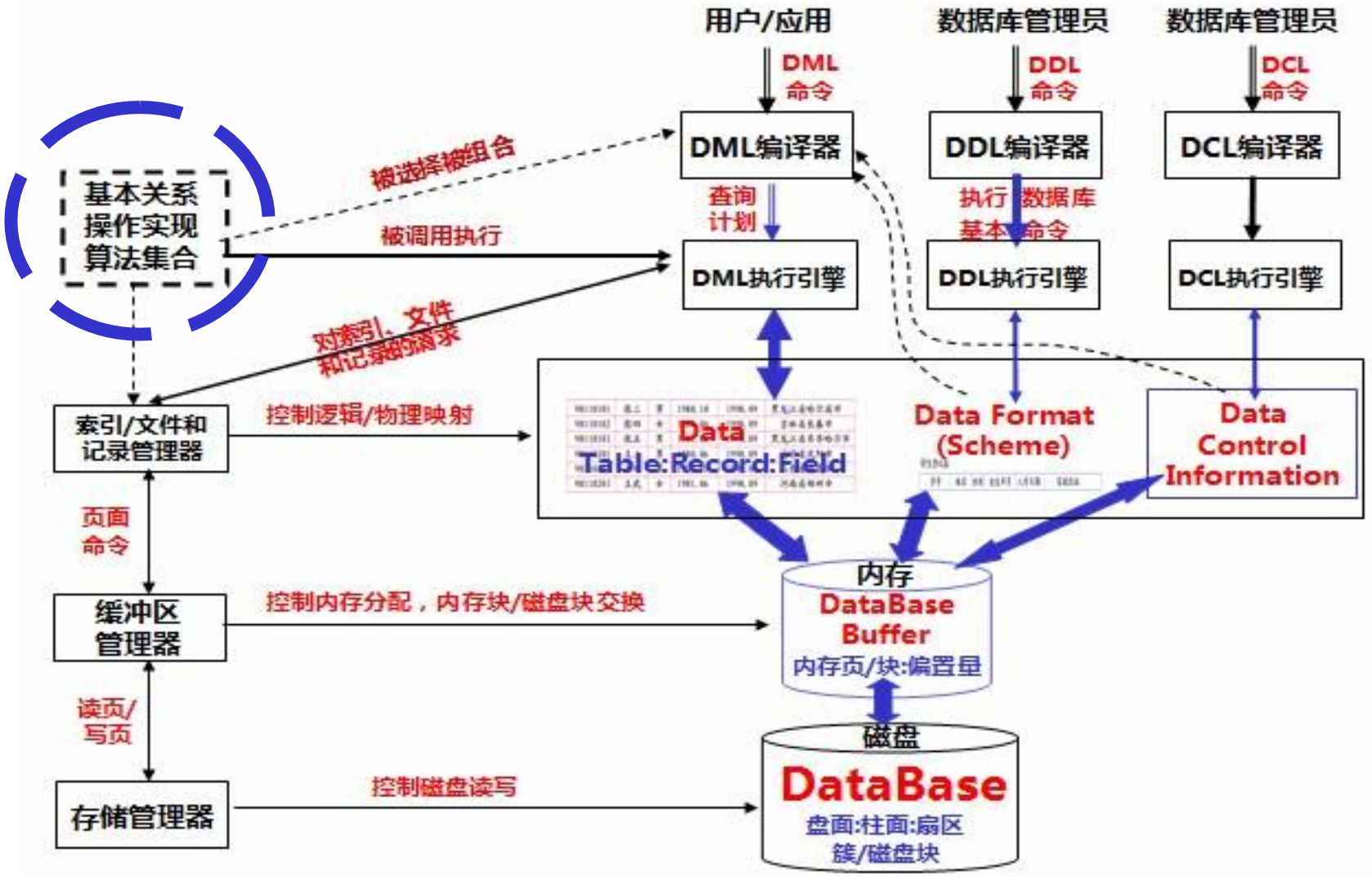
19.3 利用迭代器构造查询实现算法？

19.4 数据库查询的一趟扫描算法？

19.5 基于索引的算法？

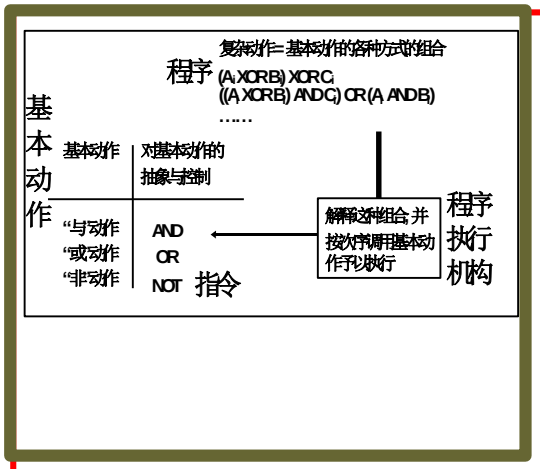
19.1 数据库查询实现算法概述

(1)“查询实现”在数据库管理系统中的位置



19.1 数据库查询实现算法概述

(2)实现数据库查询的基本思想



SQL语言

```
Select  Sname  From  Student, SC
Where  Student.S# = SC.S#  and  SC.C# =  '001'
Order By  Score  DESC;
```

程序

$$\Pi_{Sname}(\sigma_{student.s\#=sc.s\#}(Student \times SC))$$

复杂动作 = 基本动作的各种方式的组合

关系模型基本运算的各种组合

基本动作

基本动作	对基本动作的抽象与控制
"并"动作	∪
"差"动作	−
"积"动作	×
"选择"动作	σ
"投影"动作	Π

指令

解释这种组合，并按次序调用基本动作予以执行

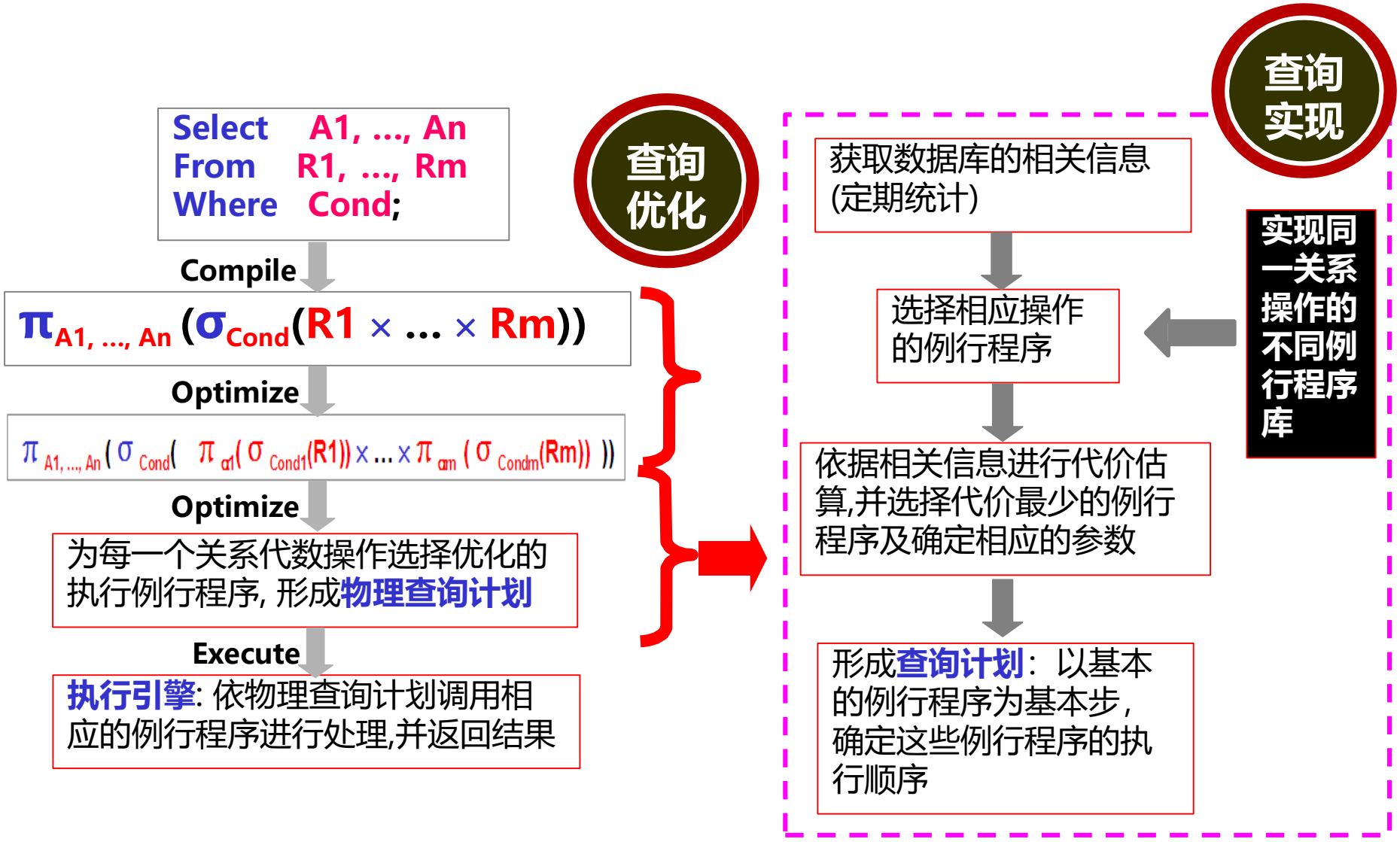
程序执行机构

关系模型基本运算

数据库管理系统

19.1 数据库查询实现算法概述

(3)查询实现vs. 查询优化



19.1 数据库查询实现算法概述

(4)查询实现算法总览

数据库的三大类操作

- 一次单一元组的一元操作
  - ✓ $\sigma_F(R)$, $\pi_\alpha(R)$ ---SELECTION, PROJECTION

迭代器算法

- 整个关系的一元操作
  - ✓ $\delta(R)$, $\gamma(R)$, $\tau(R)$ ---DISTINCT, GROUP BY, SORTING
- 整个关系的二元操作
  - ✓ 集合上的操作：$\cup_S$, $\cap_S$, $-_S$
  - ✓ 包上的操作：$\cup_B$, $\cap_B$, $-_B$
  - ✓ 积，连接：PRODUCT， JOIN

一趟扫描算法　两趟扫描算法　多趟扫描算法

基于排序的算法　基于散列的算法　基于索引的算法

第19讲 数据库查询实现算法-I

19.1 数据库查询实现算法概述?

19.2 连接操作的实现算法--由逻辑层面到物理层面?

-----由逻辑层面到物理层面

19.3 利用迭代器构造查询实现算法？

19.4 数据库查询的一趟扫描算法？

19.5 基于索引的算法？

19.2 连接操作的实现算法--由逻辑层面到物理层面

(1)连接操作的逻辑实现算法

$$R \bowtie S$$
$$R.A\ \theta\ S.B$$

T_R：关系R的元组数目；

T_S：关系S的元组数目；

[连接操作的逻辑实现算法--P0]

```
For  i = 1 to T_R
    read i-th record of R;
    For j = 1 to T_S
        read j-th record of S;
        if R.A θ S.B then
        { 串接 i-th record of R 和 j-th record of S;
          存入结果关系; }
    Next j
Next i
```

这只是逻辑算法,物理上复杂得多！

R

A	B
a	1
b	2

S

H	C
1	x
1	y
3	z

$R \underset{B=H}{\bowtie} S$

A	B	H	C
a	1	1	x
a	1	1	y

19.2 连接操作的实现算法--由逻辑层面到物理层面

(2)关系的物理存储相关的参数

物理算法需要考虑

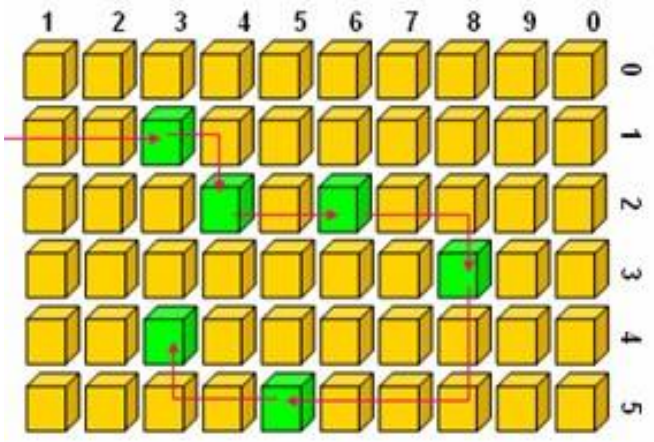
关系是存储在磁盘上的，磁盘是以磁盘块为操作单位，首先要被装载进内存(I/O操作)，然后再进行元组的处理

- T_R：关系R的元组数目；
- B_R：关系R的磁盘块数目；
- M：主存缓冲区的页数(主存每页容量等于一个磁盘块的容量)；
- l_R：关系R的每个元组的字节数；
- b：每个磁盘块的字节数；

$$B_{RxS} = T_R T_S (l_R + l_S)/b。$$

R	
A	B
a	1
b	2

S	
H	C
1	x
1	y
3	z



磁盘块为IO基本单位

19.2 连接操作的实现算法--由逻辑层面到物理层面

(3)连接操作的基本实现算法

[连接操作的基本实现算法P1]

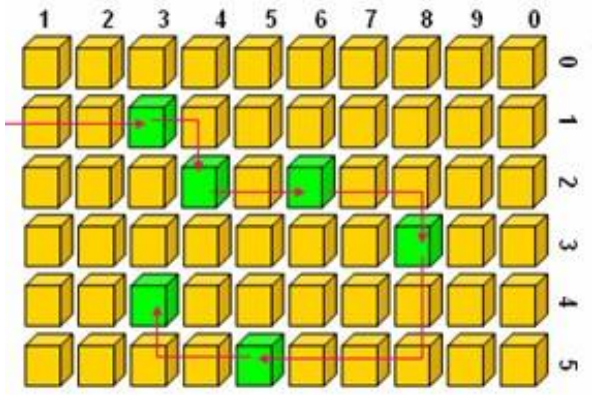
For i = 1 to B_R
  read i-th block of R;
  For j = 1 to B_S
    read j-th block of S;
    For p = 1 to b/l_R
      read p-th record of R;
      For q = 1 to b/IS
      read q-th record of S;
      if R.A θ S.B then
      { 串接 p-th record of R和q-th record of S;存入结果关系; }
    Next q
  Next p
Next j
Next i

磁盘I/O操作

内存操作

$R \bowtie S$

R.A θ S.B



定长记录块

记录1	记录2	记录3	
记录11	记录12	记录13	
记录31	记录32	记录33	

算法复杂性： I/O次数估计为 $B_R + B_R \times B_S$ (暂忽略保存结果关系的I/O次数)

应用条件： 仅需要三个内存页即可应用，一页装入R，一页装入S，一页输出。

19.2 连接操作的实现算法--由逻辑层面到物理层面

(4)连接操作的全主存实现算法

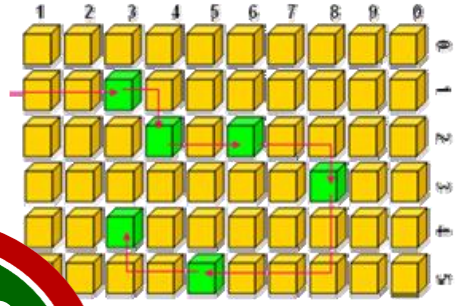
[连接操作的全主存实现算法P2]

应用条件：算法假定M >= B_R + B_S。

```
For  i = 1 to B_R //注：有可能一次性读入连续的多块
    read i-th block of R;
Next I
For j = 1 to B_S //注：有可能一次性读入连续的多块
    read j-th block of S;
Next j
For  p = 1 to T_R
    read p-th record of R;   For
    q = 1 to T_S
        read q-th record of S;
        if R.A θ S.B then
        { 串接 p-th record of R和q-th record of S;
            存入结果关系；}
    Next q
Next p
```

$R \bowtie S$

R.A θ S.B



降低I/O操作次数是关键

充分利用内存

内存如能装下两个关系

算法复杂性：I/O次数估计为B_R + B_S (暂忽略结果关系保存的I/O次数)

19.2 连接操作的实现算法--由逻辑层面到物理层面

(5)连接操作的半主存实现算法

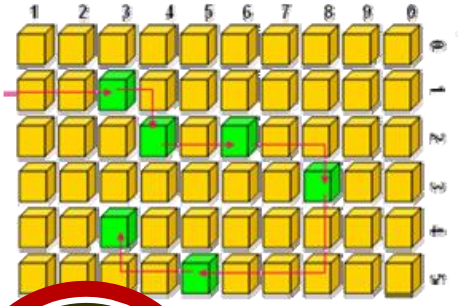
[连接操作的半主存实现算法P3]

应用条件：算法假定$B_S >= B_R$，$B_R < M$。

```
For  i = 1 to B_R //注：有可能一次性读入连续的多块
     read i-th block of R;
Next i
For j = 1 to B_S  //注：一次读入一块
     read j-th block of S;
     For  p = 1 to T_R
        read p-th record of R;
        For q = 1 to b/l_S
           read q-th record of S;
           if R.A θ S.B then
          { 串接 p-th record of R和q-th record of S;
              存入结果关系；}
        Next q
     Next p
Next j
```

$R \bowtie S$

R.A θ S.B



充分利用内存

内存仅能装下一个关系

算法复杂性： I/O次数估计为 $B_R + B_S$ (暂忽略结果关系保存的I/O次数)

19.2 连接操作的实现算法--由逻辑层面到物理层面

(6)连接操作的大关系实现算法

[连接操作的大关系实现算法P4]

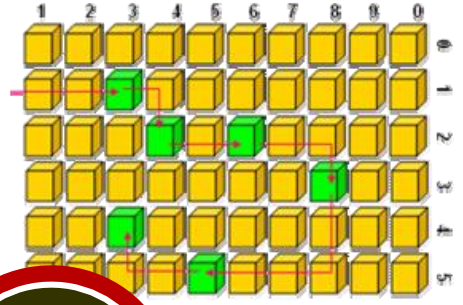
应用条件： 算法假定$B_S >= M$，$B_R >= M$。

把关系S划分为$B_S/(M-2)$个子集合，每个子集合具有M-2块。令M_S为M-2块容量的主存缓冲区，M_R为1块容量的R的主存缓冲区，还有1块作为输出缓冲区。

```
For  i = 1 to B_S/(M-2) //注：一次读入M-2块
    read i-th Sub-set of S into M_S;
    For j = 1 to B_R  //注：一次读入一块
        read j-th block of R into M_R;
        For  p = 1 to (M-2)b/l_S
            read p-th record of S;
            For q = 1 to b/l_R
                read q-th record of R;
                if R.A θ S.B then
              { 串接 p-th record of S and q-th record of R;
                存入结果关系；}
            Next q
        Next p
    Next j
Next i
```

$R \bowtie S$

R.A θ S.B



充分利用内存

两个关系都不能完全装入内存

算法复杂性： I/O次数估计为$B_R(B_S/(M-2)) + B_S$（暂忽略结果关系保存的I/O次数）

19.2 连接操作的实现算法--由逻辑层面到物理层面

(7)连接操作的其他实现算法

[连接操作的物理实现算法]

□表空间扫描法

✓基本实现算法P1

➢适用于任何情况：3块内存即可，但算法复杂性高: $B_R + B_R \times B_S$

✓全主存实现算法P2

➢要求内存能够完全装载两个关系。算法复杂性低： $B_R + B_S$

✓半主存实现算法P3

➢要求内存能够完全装载一个关系。算法复杂性低： $B_R + B_S$

✓大关系实现算法P4

➢适用于任何情况，尤其是大关系情况下比算法P1好。

➢算法复杂性低： $B_R(B_S/(M-2)) + B_S$

□归并排序(Sort-Merge)连接算法P5

□散列连接(Hash连接)算法P6

□索引连接算法 P7

第19讲 数据库查询实现算法-I

19.1 数据库查询实现算法概述?

19.2 连接操作的实现算法--由逻辑层面到物理层面?

19.3 利用迭代器构造查询实现算法？

19.4 数据库查询的一趟扫描算法？

19.5 基于索引的算法？

19.3 利用迭代器构造查询实现算法

(1)迭代器算法的提出

数据库的三大类操作

口一次单一元组的一元操作

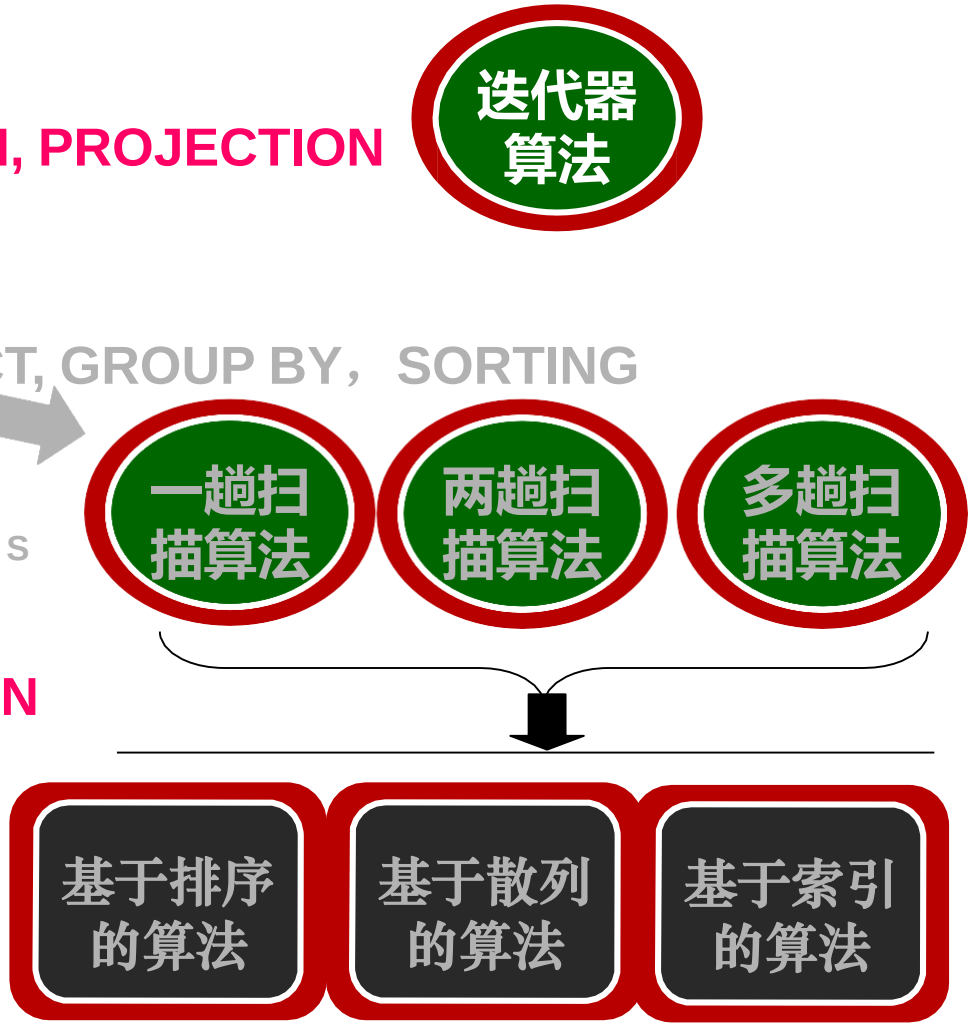
- ✓ $\sigma_F(R)$, $\pi_\alpha(R)$ ---SELECTION, PROJECTION

口整个关系的一元操作

- ✓ $\delta(R)$, $\gamma(R)$, $\tau(R)$ ---DISTINCT, GROUP BY, SORTING

口整个关系的二元操作

- ✓ 集合上的操作：$\cup_S$, $\cap_S$, $-_S$
- ✓ **包上的操作：** $\cup_B$, $\cap_B$, $-_B$
- ✓ 积，**连接：** PRODUCT， JOIN



19.3 利用迭代器构造查询实现算法

(1)迭代器算法的提出

查询实现的两种策略

$$\pi_{S\#,Sname}(\sigma_{C\#=\text{“001”} \wedge Student.S\# = SC.S\#}(Student \times SC))$$

- **物化计算策略**

$$Temp1 \leftarrow Student \times SC$$

$$Temp2 \leftarrow \sigma_{C\#=\text{“001”} \wedge Student.S\# = SC.S\#}(Temp1)$$

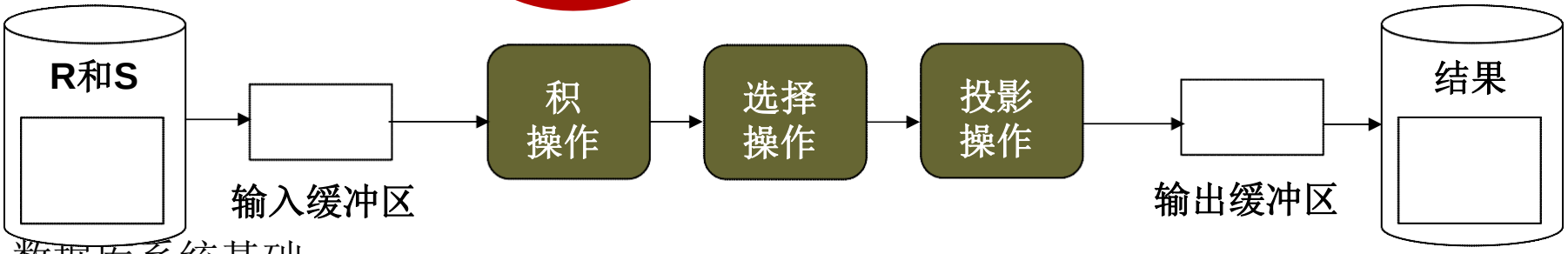
$$\text{结果关系} \leftarrow \pi_{S\#,Sname}(Temp2)$$

是一个关系操作还是一组关系操作相当于扫描一遍数据库

中间结果是否完整的存储(可存于内存或外存中)

- **流水线计算策略**

迭代器算法



19.3 利用迭代器构造查询实现算法

(2)迭代器算法的基础

迭代器：迭代的读取一个集合中的每一个元素，而封装其读取细节

面向对象的构造--可学习《面向对象程序设计语言》如C++,Java

- **类　Class A { }**
- **类的函数/方法　Class A { void f1(){ }　}**
- **类的继承　Class　A : B**
- **类的实例化--对象：例如“ a = new A；”或者“A　a”;**

有一个抽象类

```
class iterator  { void Open(); tuple
   GetNext(); void Close();
   iterator &inputs[];
}
```

```
R.Open();
t := R.GetNext();
R.Close();
```

所有关系操作可继承此迭代器进行构造。

不同操作，可以构造不同的**Open(),GetNext(), Close()**函数

19.3 利用迭代器构造查询实现算法

(3)迭代器的构造

迭代器示例：

表空间扫描法—读取关系

R

```
Open() {
      b := R的第一块;
      t  := b的第一个元组;
}

GetNext() {
   IF ( t 已超过块b的最后一个元组 ) {
         将b前进到下一块
         IF (没有下一块)
               RETURN  NotFound;
         ELSE  /* b是一个新块 */
               t  := b的第一个元组;
      }
     oldt := t; 将t前进到b的
     下一元组;
     RETURN oldt;
}

Close() {
}
```

19.3 利用迭代器构造查询实现算法

(3)迭代器的构造

迭代器示例：

R∪S的算法

Union(R,S)

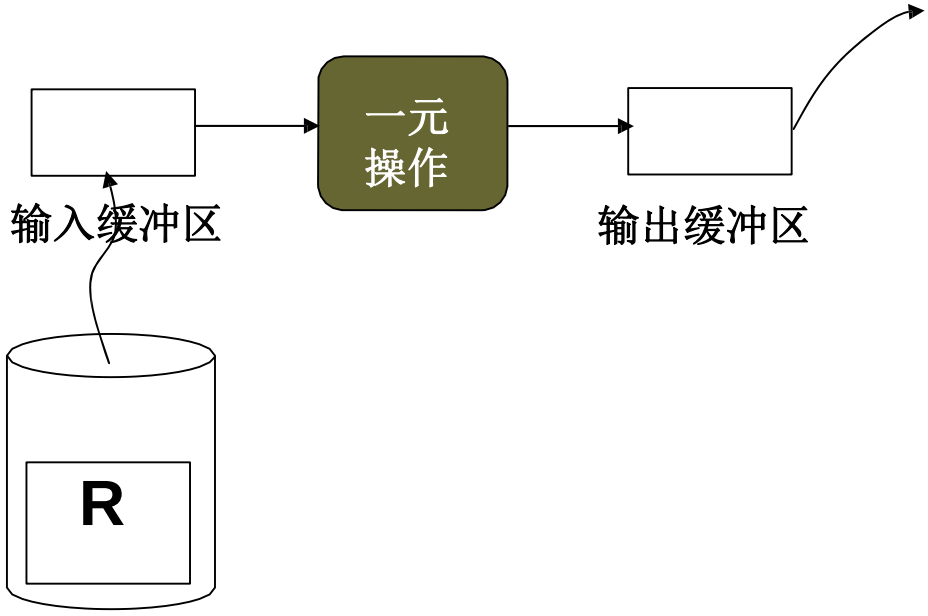
```
Open() {
    R.Open();
    CurRel := R;
}
GetNext() {
   IF ( CurRel == R ）{
        t:= R.GetNext();
        IF (t<> NotFound)
            RETURN t; /* 未处理完 */
        ELSE { /* 已处理完R */
            S.Open();
            CurRel := S;
        }
    }
    RETURN S.GetNext();
    /* S处理完返NotFound,其将NotFound再返回 */
}
Close() {
    R.Close();
    S.Close();
}
```

19.3 利用迭代器构造查询实现算法

(3)迭代器的构造

迭代器示例： **SELECTION(R)**

$\sigma_F(R)$, $\pi_\alpha(R)$
---SELECTION, PROJECTION
- 一次一个元组的操作
- 最少只需一个Block的内存空间



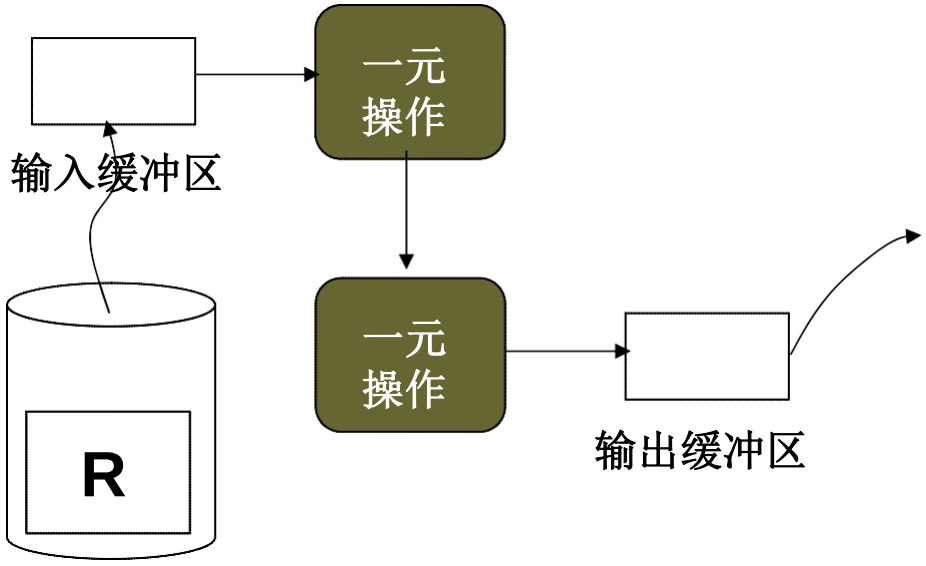

```
Open() {
    R.Open();
}
GetNext() {
     Cont:
     t:=R.GetNext();
     IF (t<> NotFound)
          IF F(t) == TRUE
               RETURN t;
          ELSE  GOTO  Cont;
     ELSE  RETURN NotFound;
}
Close() {
    R.Close();
}
```

19.3 利用迭代器构造查询实现算法

(3)迭代器的构造

迭代器示例：

$\pi_{\alpha}(\sigma_{F}(R))$



PROJECTION(SELECTION(R))

```
Open() {
    SELECTION.Open();
}
GetNext() {
    t:= SELECTION.GetNext();
    IF (t<> NotFound) {
        p := PROJECTION(t, α)
        RETURN p;
    }
    ELSE  RETURN NotFound;
}
Close() {
    SELECTION.Close();
}
```

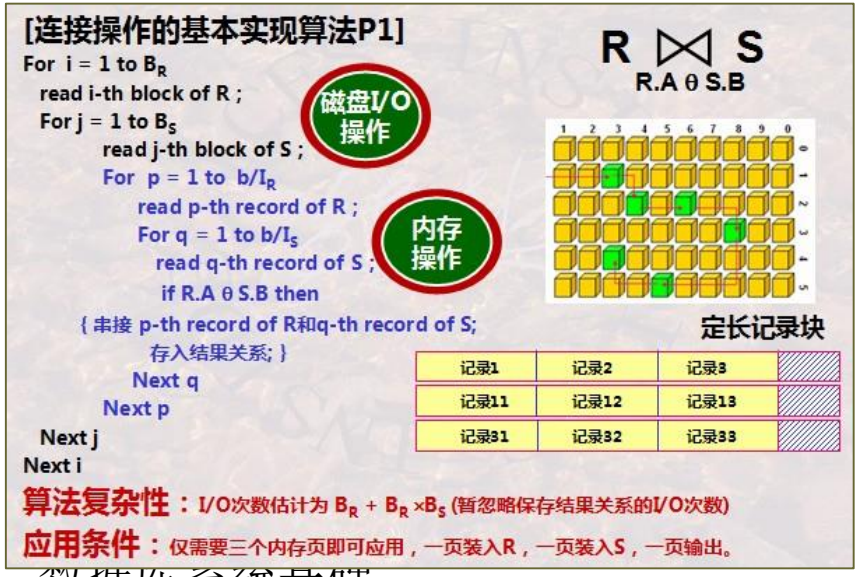
19.3 利用迭代器构造查询实现算法

(3)迭代器的构造

迭代器示例： **Join(R,S)**

R Join S

```
Open() {
    R.Open();    S.Open();
    r:= R.GetNext();
}
GetNext() {
      REPEAT{
      s:= S.GetNext();
      IF ( s == NotFound ) {
         S.Close();
         r: = R.GetNext();
         IF (r == NotFound)
               RETURN NotFound;
         }ELSE { S.Open();
               s := S.GetNext(); }
    UNTIL (r 与s能够连接);
    RETURN  r和s的连接;
} }
Close() {
    R.Close();      S.Close();
}
```



第19讲 数据库查询实现算法-I

19.1 数据库查询实现算法概述?

19.2 连接操作的实现算法--由逻辑层面到物理层面?

19.3 利用迭代器构造查询实现算法？

19.4 数据库查询的一趟扫描算法？

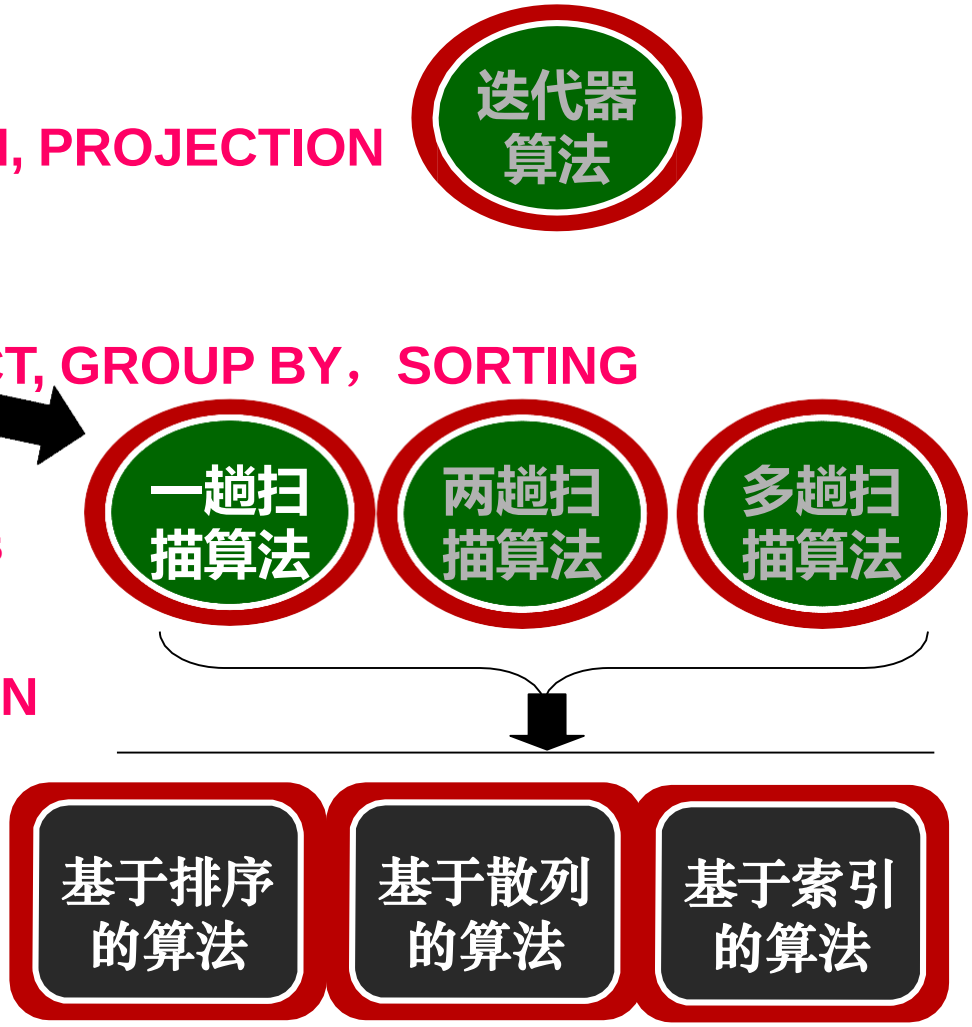
19.5 基于索引的算法？

19.4 数据库查询的一趟扫描算法

(1)什么是一趟算法?

数据库的三大类操作

- 一次单一元组的一元操作
  - ✓ $\sigma_F(R)$, $\pi_\alpha(R)$ ---SELECTION, PROJECTION
- 整个关系的一元操作
  - ✓ $\delta(R)$, $\gamma(R)$, $\tau(R)$ ---DISTINCT, GROUP BY，SORTING
- 整个关系的二元操作
  - ✓ 集合上的操作：$\cup_S$ ，$\cap_S$，$-_S$
  - ✓ 包上的操作：$\cup_B$ ，$\cap_B$，$-_B$
  - ✓ 积，连接：PRODUCT， JOIN



19.4 数据库查询的一趟扫描算法

(2)关系/表数据的读取算法

关系/表数据的读取---完整地读取一个关系

■**聚簇关系**—关系的元组集中存放(一个块中仅是一个关系中的元组):

TableScan(R) ---表空间扫描算法

✓扫描结果未排序: **B(R)**

SortTableScan(R)

✓扫描结果排序: **3B(R)**

IndexScan(R)---索引扫描算法

✓扫描结果未排序: **B(R)**

SortIndexScan(R)

✓扫描结果排序: **B(R)** or **3B(R)**

■**非聚簇关系**—关系的元组不一定集中存放(一个块中不仅是一个关系中的元组):

✓扫描结果未排序: **T(R)**

✓扫描结果排序: **T(R) + 2B(R)**

B(R)是R的存储块数目
T(R)是R的元组数目

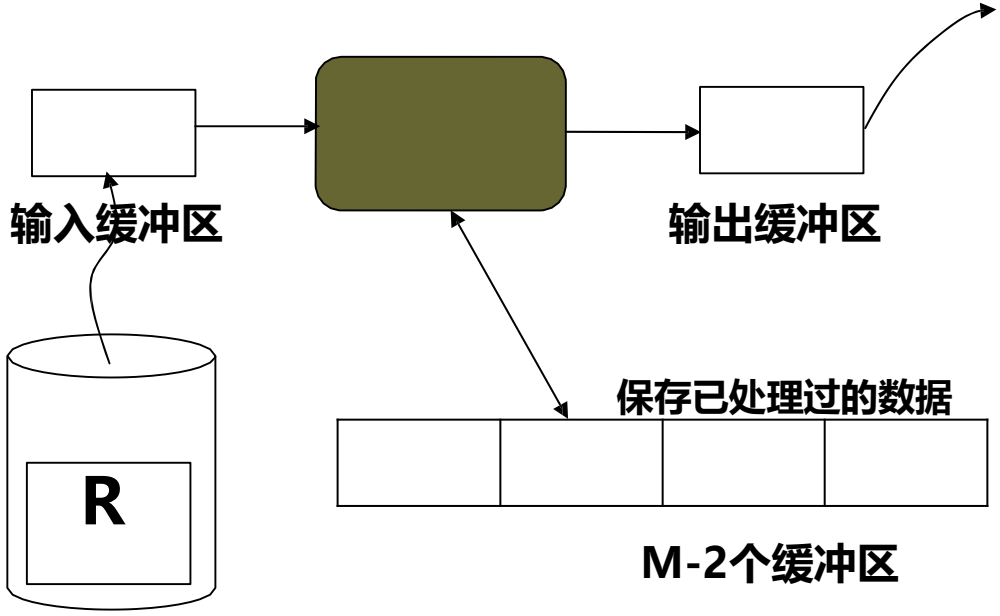
19.4 数据库查询的一趟扫描算法

(3)整个关系的一元操作实现算法

SELECT DISTINCT * FROM SC

去重复：&(R)

- 需要在内存中保存已处理过的元组
- 当新元组到达时，需与之前处理过的元组进行比较
- **建立不同的内存数据结构，来保存之前处理过的数据，以便快速处理整个关系上的操作**
- **算法复杂性**：B(R)
- **应用条件**：B(&(R))<=**M**



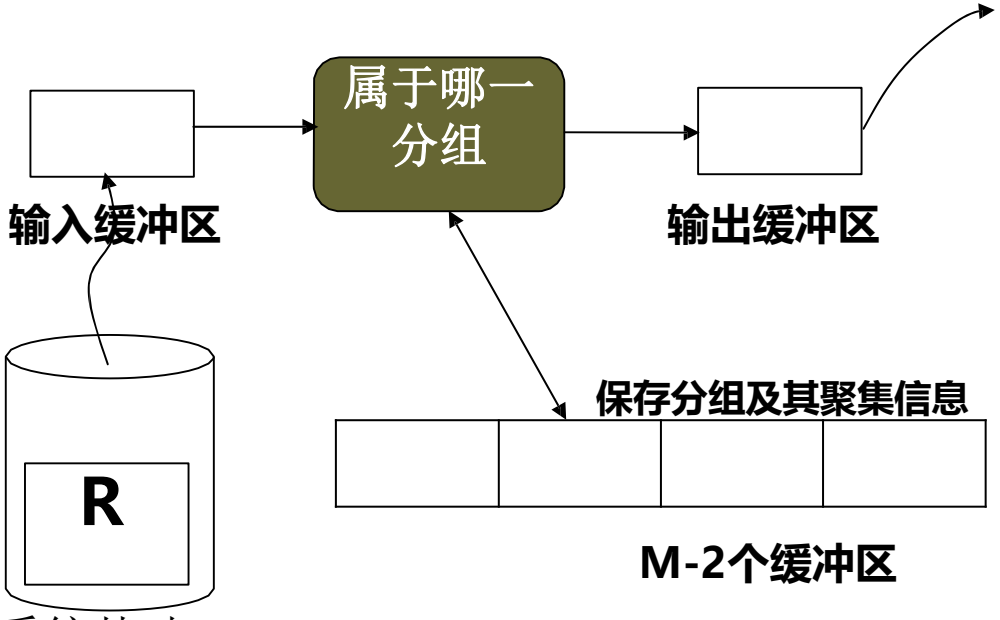
建立内存数据结构，以快速定位一个元组，如排序结构/散列结构/B+树等

(3)整个关系的一元操作实现算法

分组聚集$\gamma_L(R)$

```
SELECT S#, AVG(Score)
FROM SC
GROUP BY S#;
```

- 需要在内存中保存所有的分组
- 保存每个分组上的聚集信息
- **建立不同的内存数据结构，来保存之前处理过的数据，以便快速处理整个关系上的操作**
- **算法复杂性：B(R)**
- **应用条件：所有分组的数量应能在内存中完整保存**

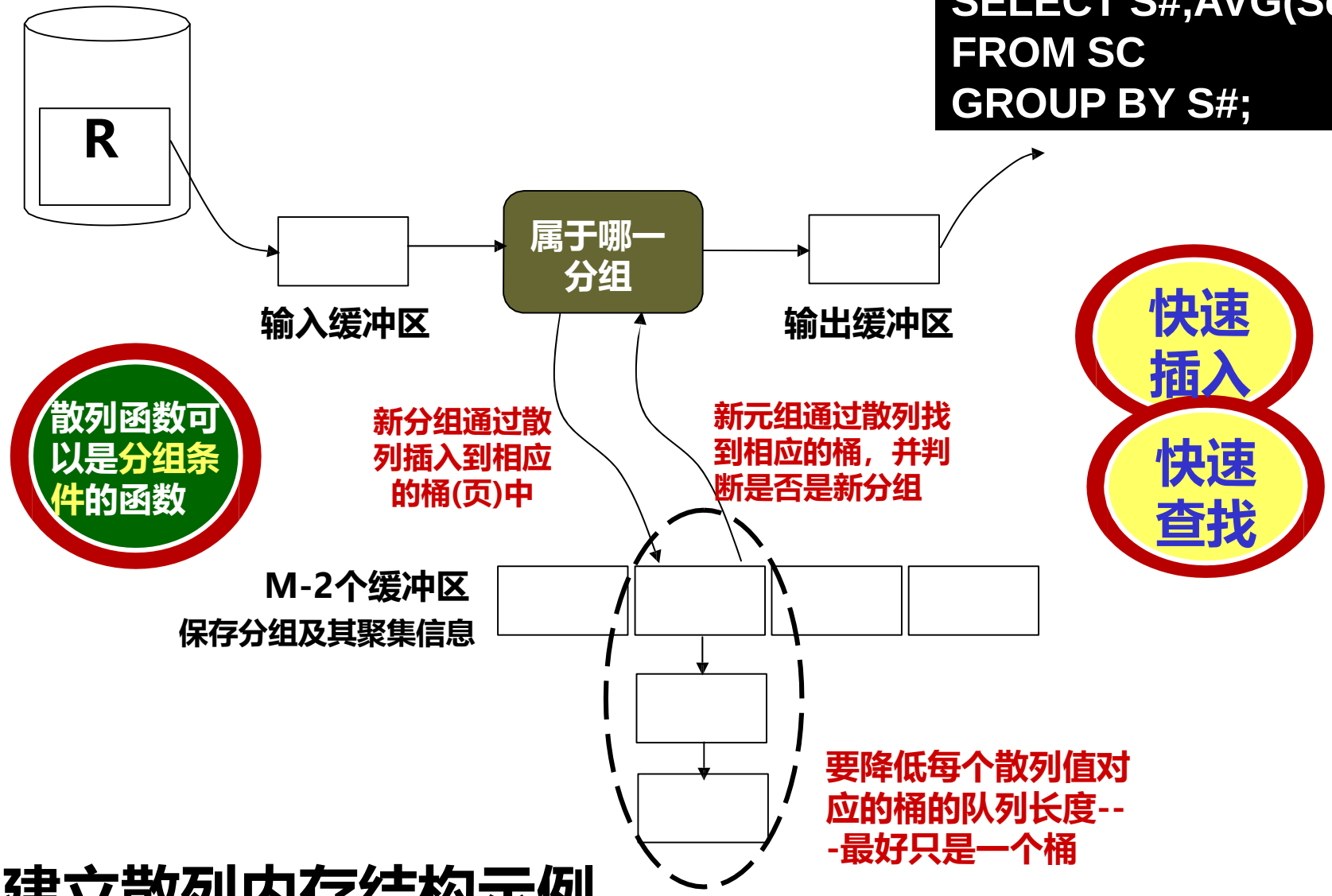


建立内存数据结构，以快速定位一个元组，如排序结构/散列结构/B+树等

19.4 数据库查询的一趟扫描算法

(3)整个关系的一元操作实现算法

```
SELECT S#,AVG(Score)
FROM SC
GROUP BY S#;
```



建立散列内存结构示例

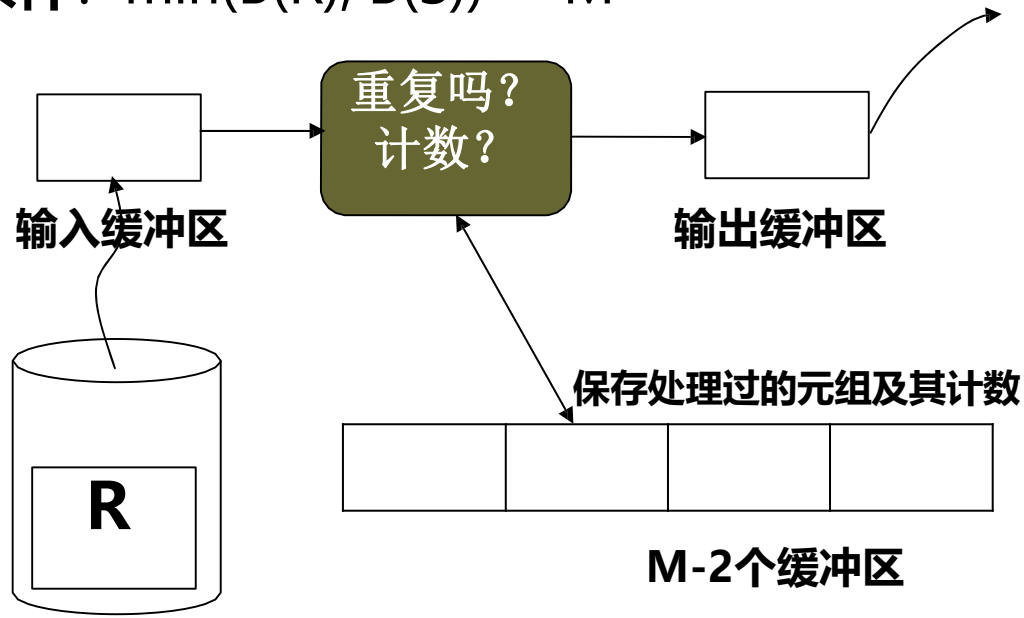
19.4 数据库查询的一趟扫描算法

(4)整个关系的二元操作实现算法

集合上的操作： $\cup_S$, $\cap_S$, $-_S$

包上的操作： $\cup_B$, $\cap_B$, $-_B$

- 扫描一个关系，然后再扫描另一个关系
- **集合**的操作需要去重复；**包**的操作需要做计数—每个元组的出现次数----具体操作还需具体分析
- **算法复杂性**：B(R)+B(S)
- **应用条件**：min(B(R), B(S))<=M



建立内存数据结构，以快速定位一个元组，如排序结构/散列结构/B+树等

19.4 数据库查询的一趟扫描算法

(4)整个关系的二元操作实现算法

[连接操作实现算法P4的改进]

- **P4 算法着重于降低I/O次数，在I/O次数不变的情况下，能否还提高性能呢?**
- **对内存操作的两重循环是否可改善呢?**

散列S的M-2个数据块，调整相应的操作，是否改善了呢?

[连接操作的大关系实现算法P4]

应用条件：算法假定$B_S >= M$，$B_R >= M$。

把关系S划分为$B_S/(M-2)$个子集合，每个子集合具有M-2块。令M_S为M-2块容量的主存缓冲区，M_R为1块容量的R的主存缓冲区，还有1块作为输出缓冲区。

```
For i = 1 to Bs/(M-2) //注：一次读入M-2块
    read i-th Sub-set of S into Ms ;
    For j = 1 to BR //注：一次读入一块
        read j-th block of R into MR ;
        For p = 1 to (M-2)b/Ls
            read p-th record of S ;
            For q = 1 to b/lR
                read q-th record of R ;
                if R.A θ S.B then
                { 串接 p-th record of S and q-th record of R ;
                  存入结果关系 ; }
            Next q
        Next p
    Next j
Next i
```

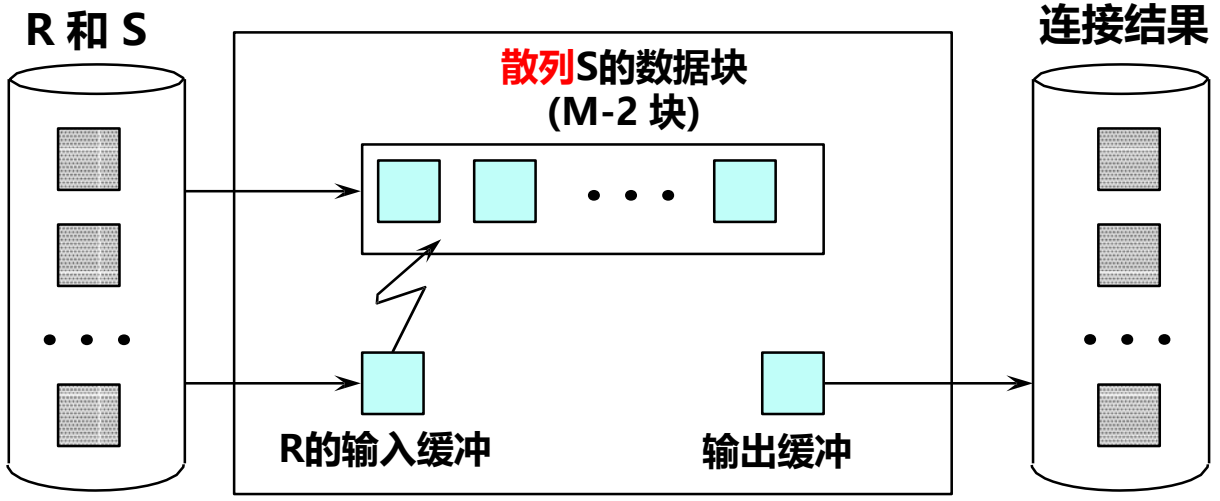
R ⋈ S, R.A θ S.B

充分利用内存

两个关系都不能完全装入内存

散列函数可取连接条件中的相应属性

如R.A θ S.B, 即S按B属性值进行散列，R按A属性值进行散列



第19讲 数据库查询实现算法-I

19.1 数据库查询实现算法概述?

19.2 连接操作的实现算法--由逻辑层面到物理层面?

19.3 利用迭代器构造查询实现算法？

19.4 数据库查询的一趟扫描算法？

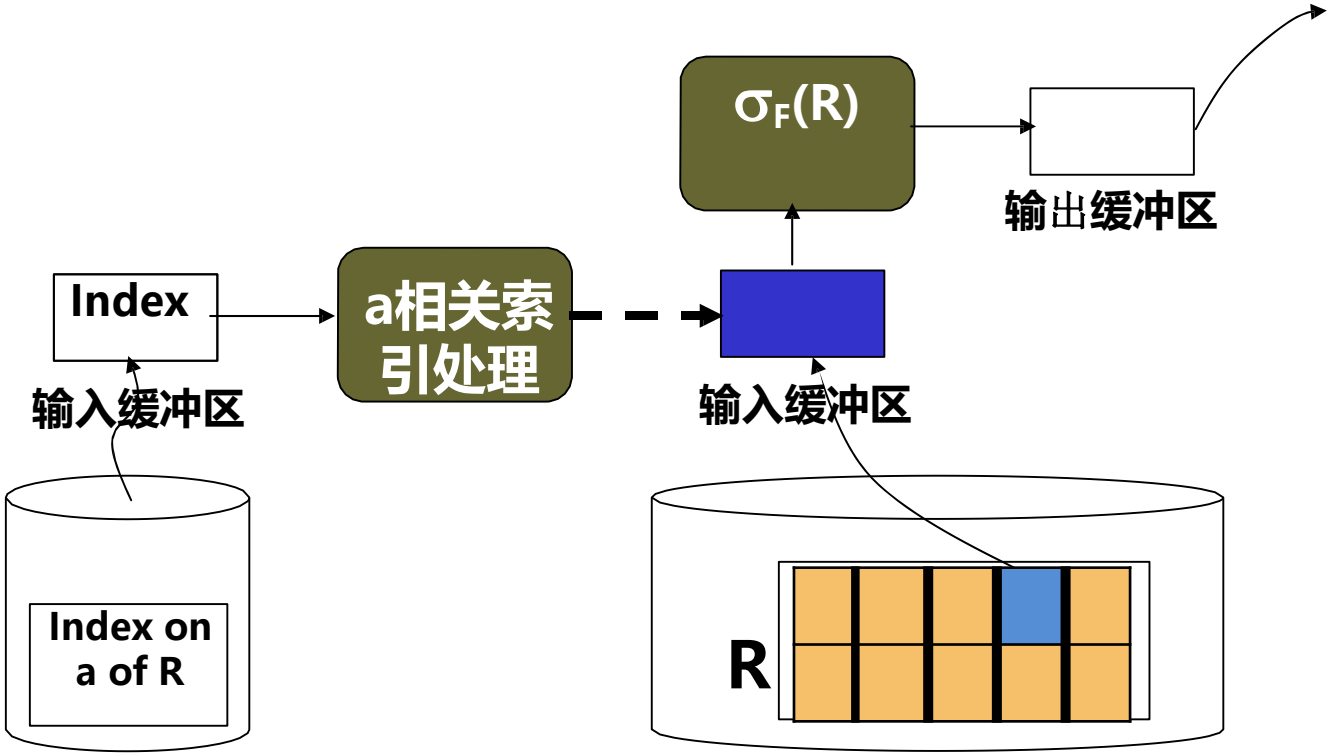
19.5 基于索引的算法？

19.5 基于索引的算法

(1)基于索引的选择算法?

$\sigma_F(R)$ ---SELECTION

- 选择条件中有涉及到索引属性时，可以使用索引，辅助快速检索；
- 在某些属性上存在着索引，可能在多个属性上都存在着索引；
- 聚簇和非聚簇索引，使用时其效率是不一样的。



19.5 基于索引的算法

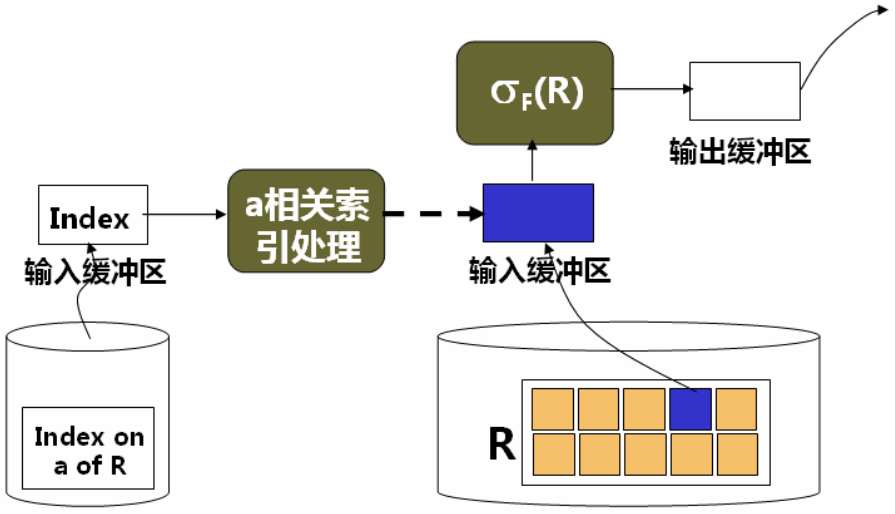
(1)基于索引的选择算法?

索引应用分析示例

假设B(R)=1000，T(R)=20000， 即：R有20000个元组存放到1000个块中。a是R的一个属性，在a上有一个索引，并且考虑$\sigma_{a=0}(R)$操作：

- 如果R是聚簇的，且不使用索引，查询代价＝1000个I/O；
- 如果R不是聚簇的，且不使用索引，查询代价＝20000个I/O 。
- 如果V(R,a)=100且索引是聚簇的，查询代价=1000/100=10个I/O。

V(R, a)表示a属性在R中出现的不同值的个数。



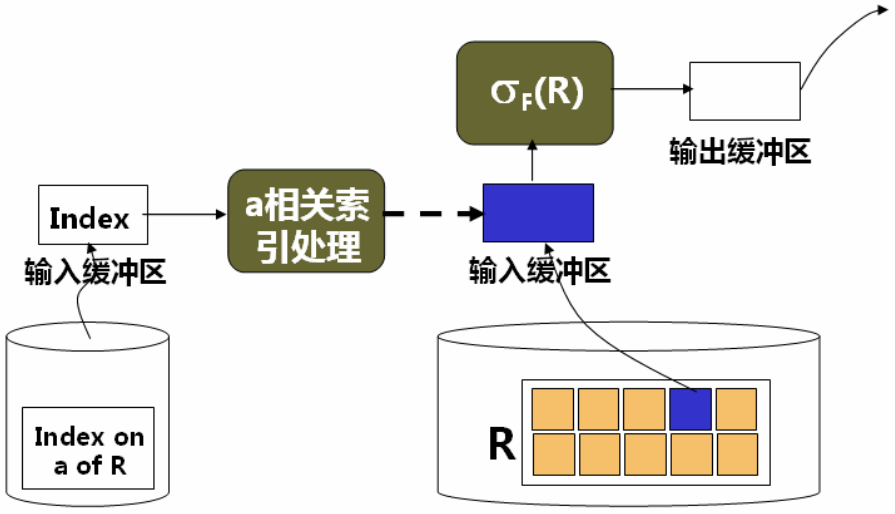
19.5 基于索引的算法

(1)基于索引的选择算法?

索引应用分析示例(续)

假设B(R)=1000，T(R)=20000， 即：R有20000个元组存放到1000个块中。a是R的一个属性，在a上有一个索引，并且考虑$\sigma_{a=0}(R)$操作：

- 如果V(R,a)=100且索引是非聚簇的，查询代价?
  =20000/100=200个 I/O。
- 如果V(R,a)=20000，即a是关键字，查询代价？
  =20000/20000=1个 I/O。不管是否是聚簇的。



19.5 基于索引的算法

(2)基于有序索引的连接算法? Zig-Zag连接算法

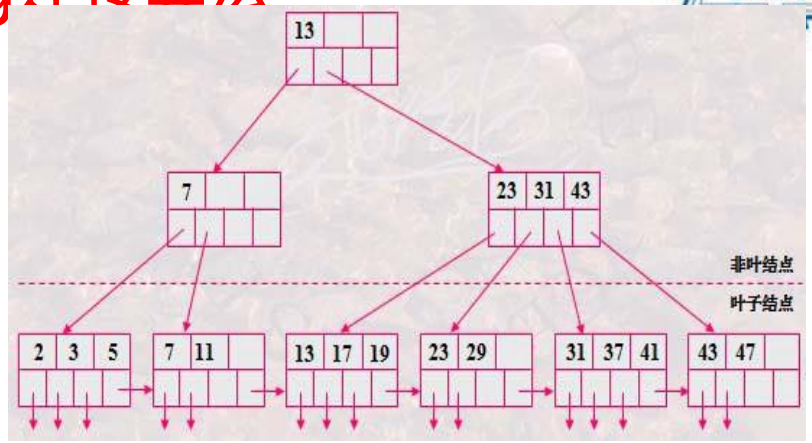
$$R \bowtie S$$

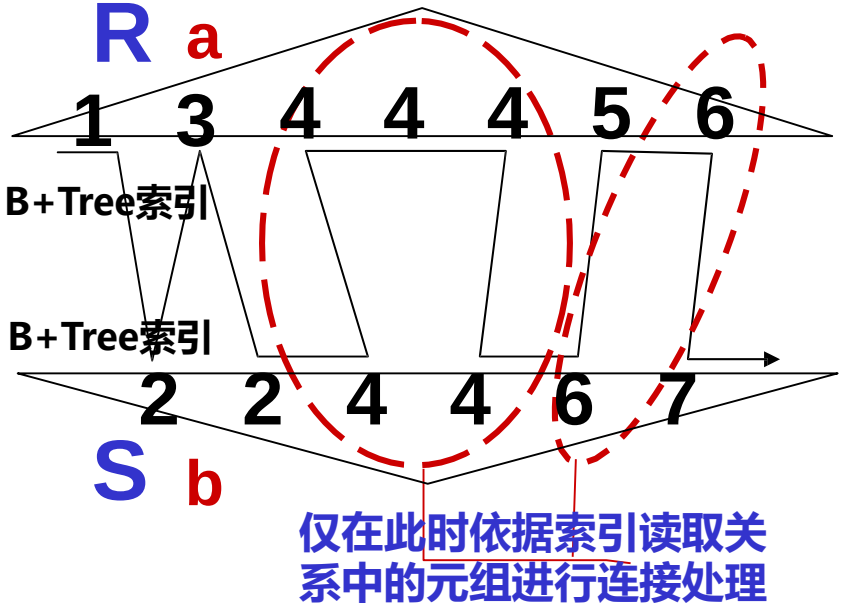
R.Y = S.Y

R 和S 都有在Y属性上的B+Tree索引；
R 和S均从左至右读取索引树的叶子结点。

(1)读R的第一个索引项赋予a，再读S的第一个索引项赋予b；
(2)如果a≠b, 则：
(21)如果a<b，则将R的下一个索引项赋予a;继续执行(2)
(22)如果a>b，则将S的下一个索引项赋予b;继续执行(2)
(3)如果a=b，则将R和S关系中对应的元组读出并进行连接，直到R的所有相等的a值 和S的所有相等的b值对应的元组都处理完毕；将R的下一个索引项赋予a,继续执行(2)。

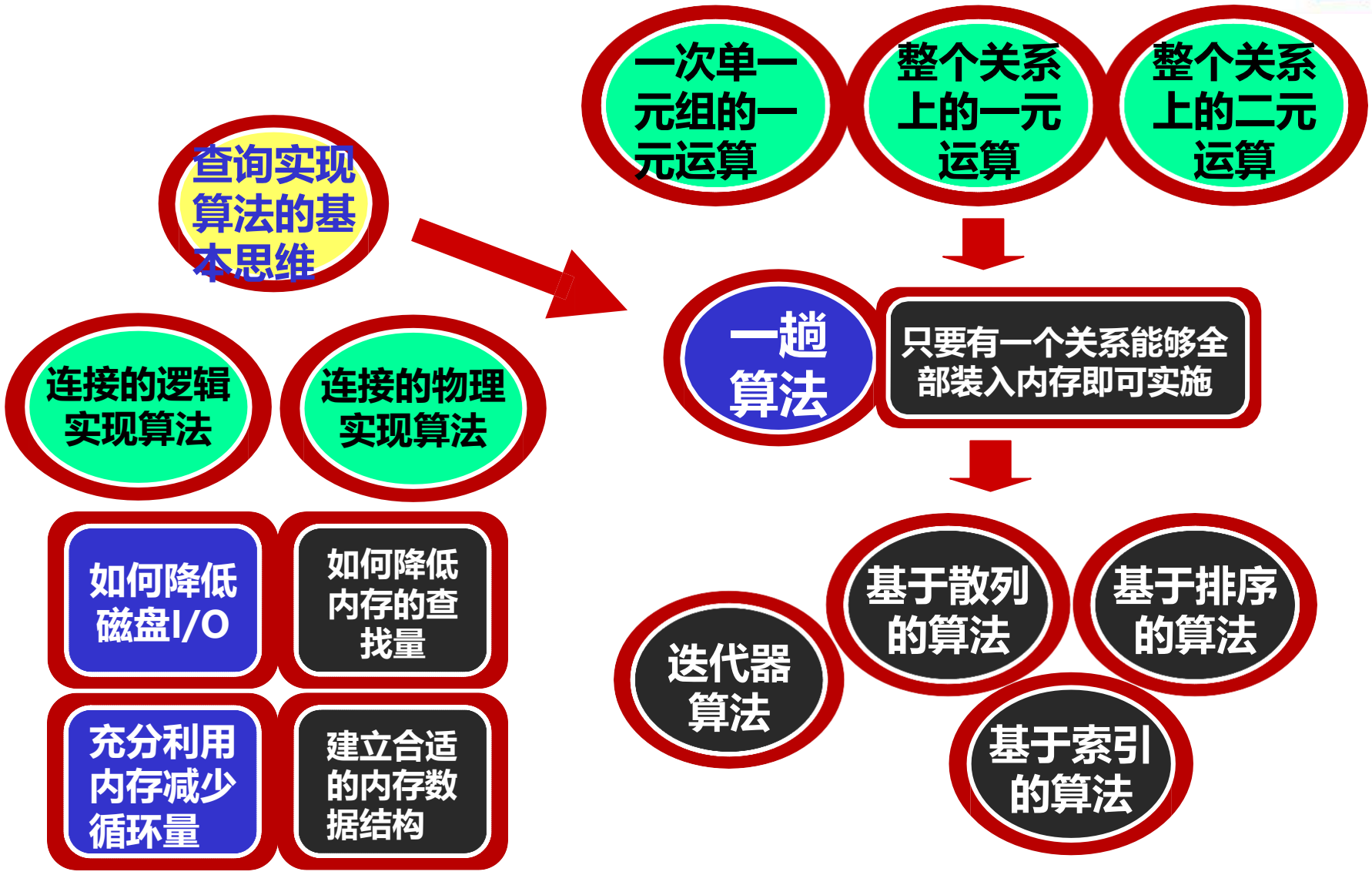
Zig-Zag算法，“跳来跳去”





回顾本讲学了什么

回顾本讲学习了什么?


查询实现算法的基本思维
一次单一元组的一元运算
整个关系上的一元运算
整个关系上的二元运算
连接的逻辑实现算法
连接的物理实现算法
一趟算法
只要有一个关系能够全部装入内存即可实施
如何降低磁盘I/O
如何降低内存的查找量
充分利用内存减少循环量
建立合适的内存数据结构
迭代器算法
基于散列的算法
基于排序的算法
基于索引的算法