第10章 查询处理

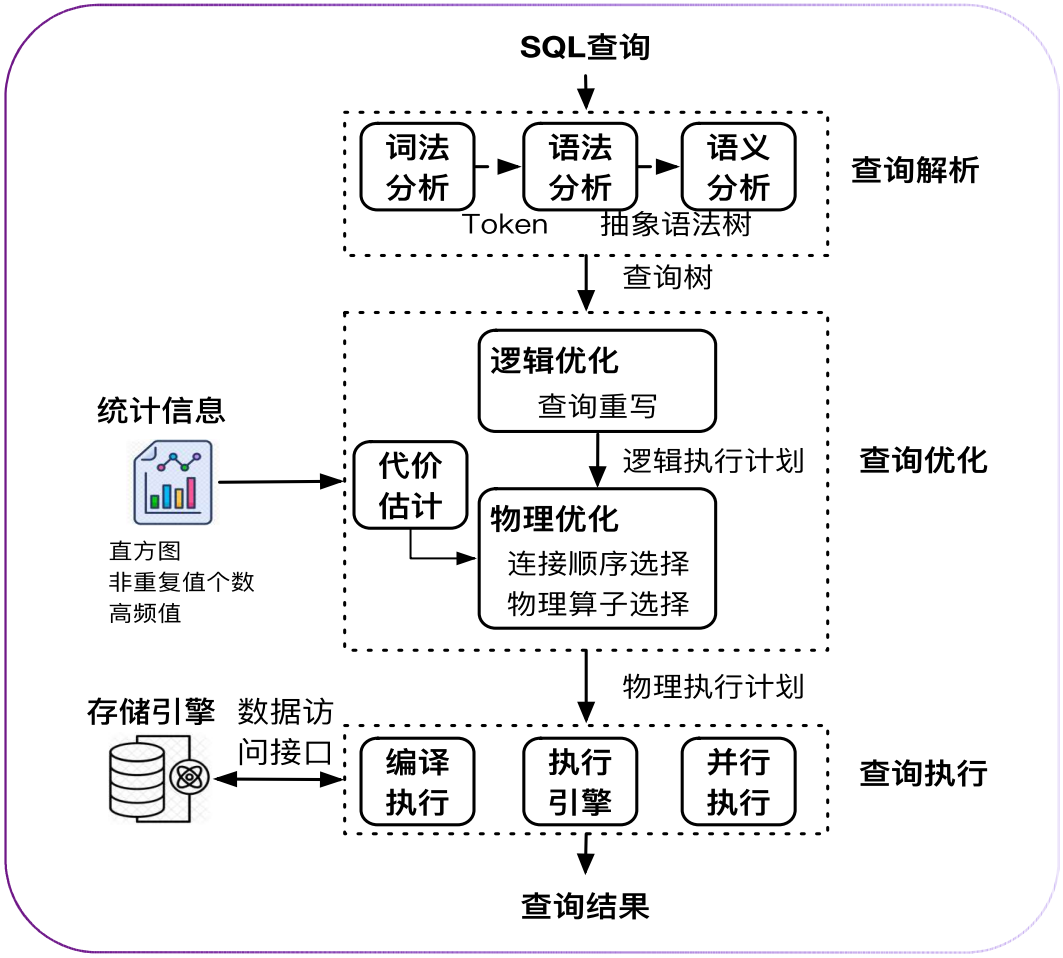
苏州大学 费子成

feizicheng@suda.edu.cn
https://web.suda.edu.cn/feizicheng/

总目录

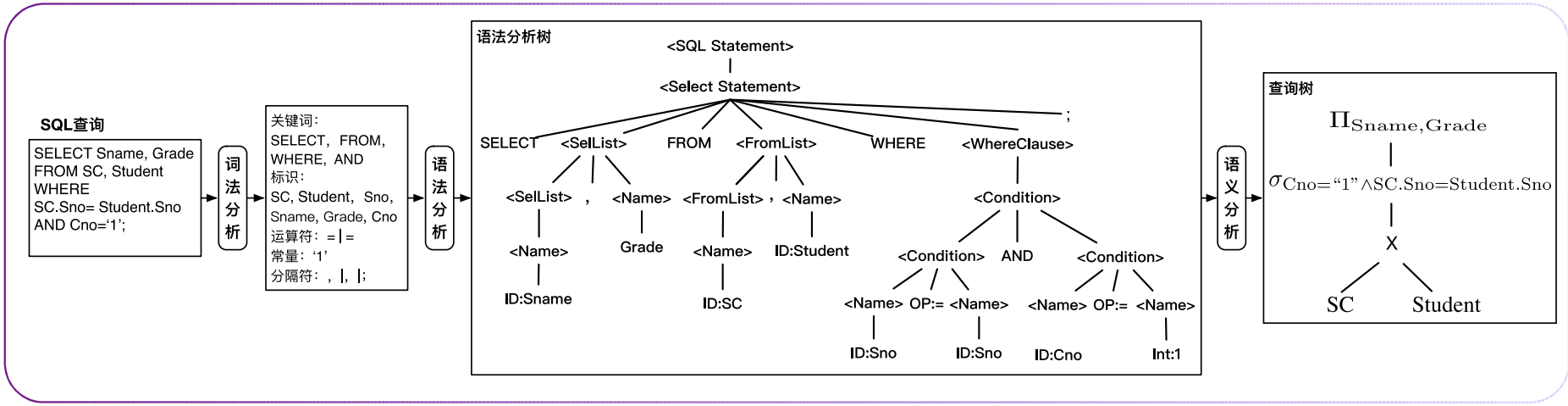
查询处理概述

- 查询处理是指针对一条查询，从数据库中提取目标数据时对该查询进行的处理过程，包括：
  - **查询解析**：将查询语句转化为容易被数据库执行的表达
  - **查询优化**：为优化查询的执行计划而进行各种逻辑等价的转换
  - **查询执行**：查询的实际执行



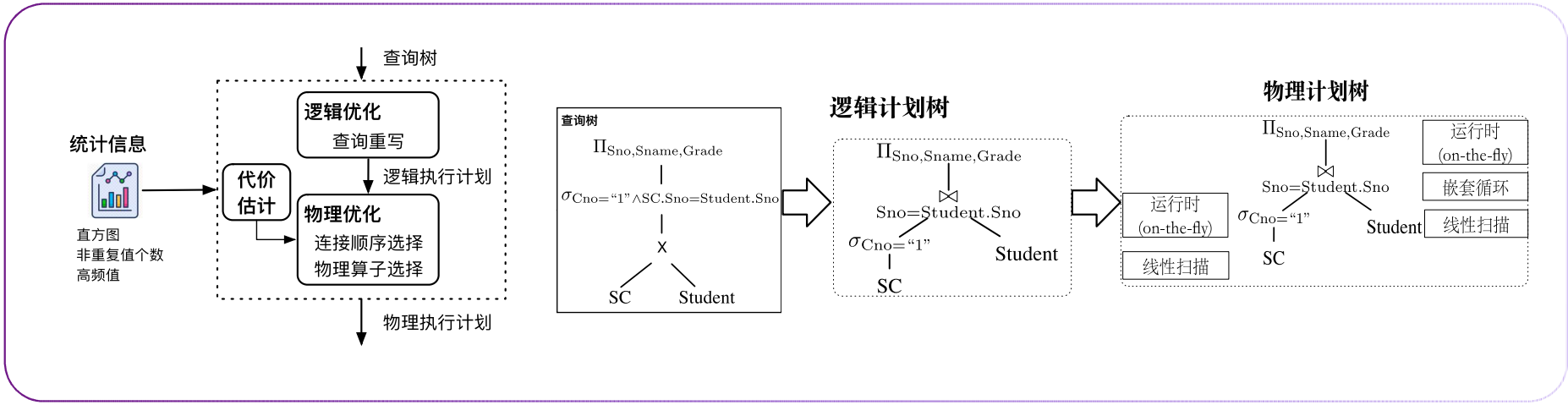
查询解析

- **输入SQL语句，输出查询树**
- **词法分析**：提取出SQL语句中的关键词、常量等成分
- **语法分析**：将词法分析提取的成分构建为一颗语法分析树
- **语义分析**：检查语法分析树的语义正确性，并将其转化为基于关系代数式表达的查询树



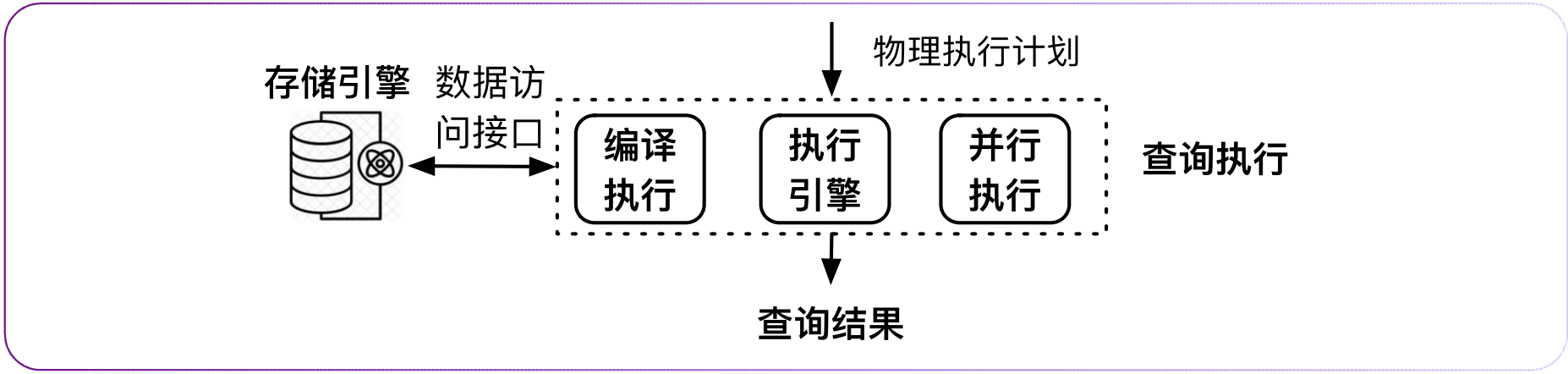
查询优化

- **输入查询树，输出物理执行计划**
- **逻辑优化：**依赖于逻辑等价变换规则将关系代数表达式变换为有更高执行效率的形式
- **代价估计：**用来预测各执行计划的资源消耗大小
- **物理优化：**依赖于代价估计器估计各执行计划的代价，以从中选取资源消耗最小的执行计划



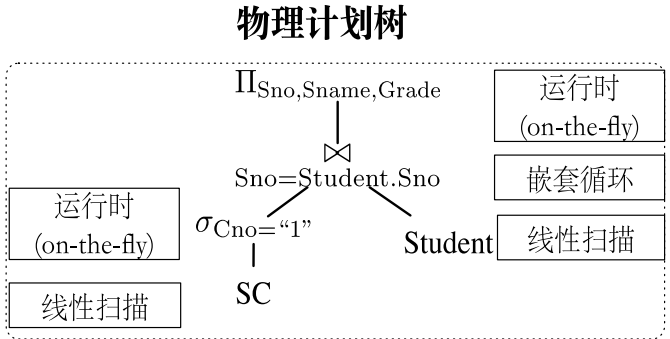
查询执行

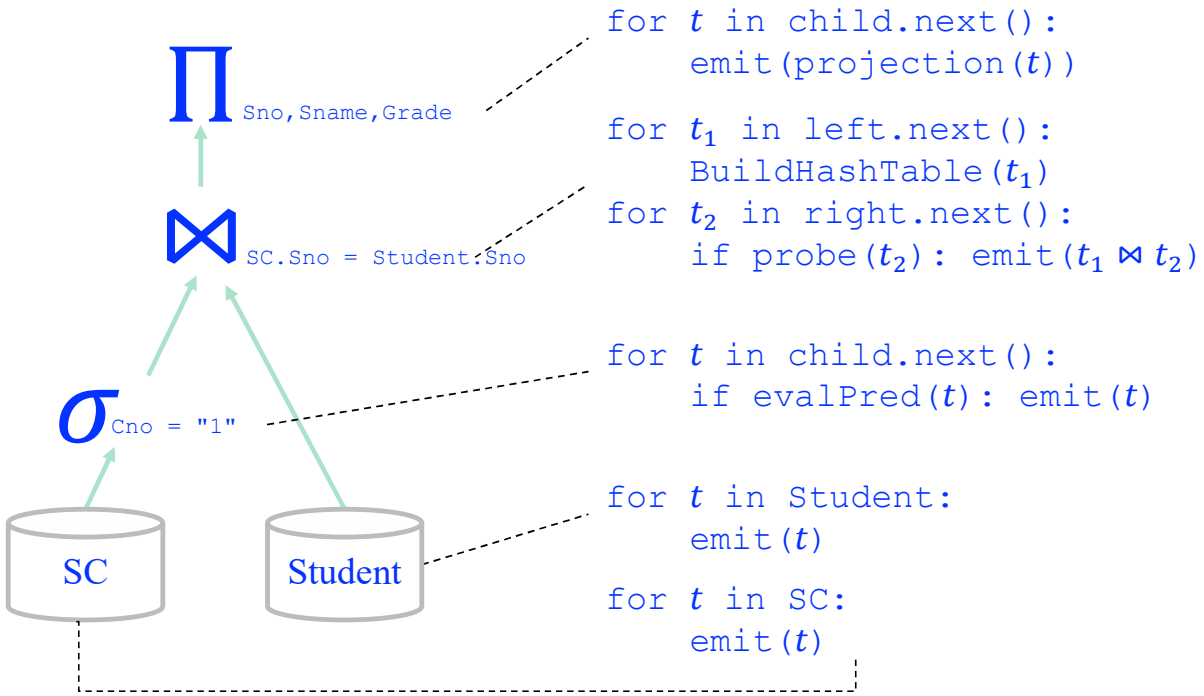
- **输入为查询的执行计划，输出为查询结果**
- **计划编译**：使用查询执行模型将物理执行计划编译为可执行代码，再将代码实际执行以返回查询结果。由于实际执行的过程中需要读取或写入数据，因此需要调用存储引擎的访问接口来访问数据
- **计划执行**：包括执行模型以及通过编译技术和并行技术来提升执行效率



查询执行

```
SELECT Sno, Sname, Grade
FROM SC, Student
WHERE SC.Sno = Student.Sno
AND Cno = "1"
```

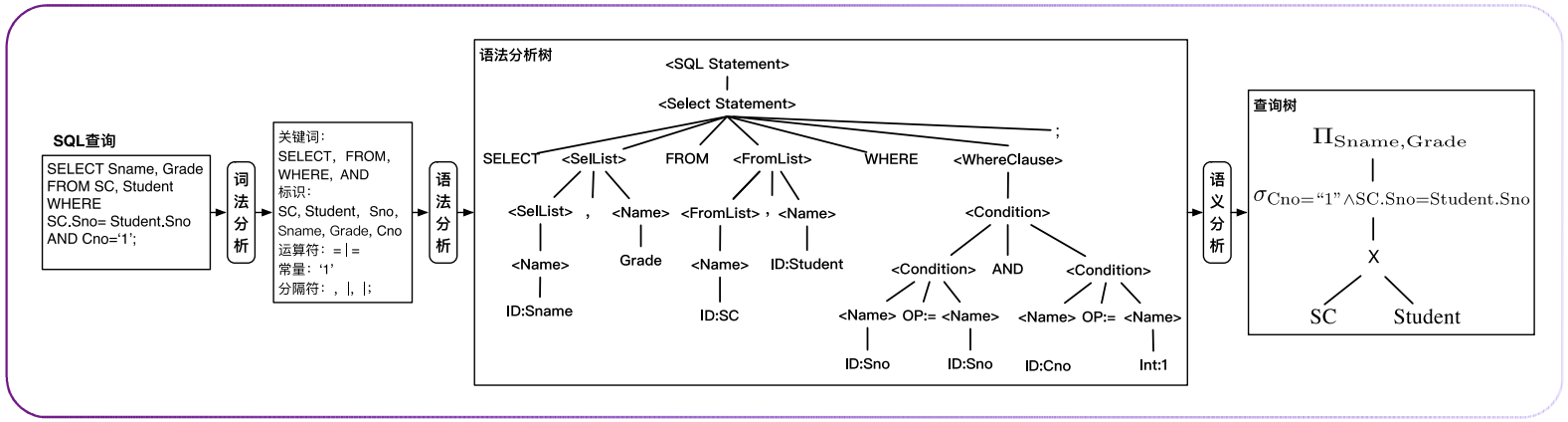




目录

SQL解析

- **查询优化开始前，数据库系统首先需要将SQL查询语句转换为等效的关系代数运算，且通常以基于关系代数的查询树来表示**
- **由三个过程构成：**
  - 词法分析：提取出SQL关键字、标识、常量
  - 语法分析：构建一颗语法分析树
  - 语义分析：进行语义正确性检查，并转化为查询树



数据库模式例子

- **为方便阐述，考虑Student和SC两个关系组成的数据库模式**
  - 以此为例介绍SQL解析的过程

Student关系

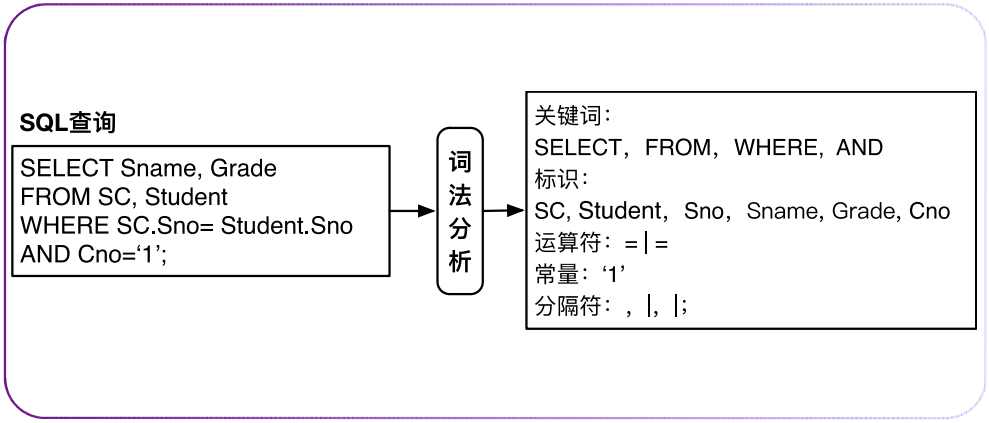
Sno (学号)	Sname (姓名)	Sgender (性别)	Sage (年龄)	Sdept (所在系)
2021310721	李博	男	17	CS
2021310722	赵宇	男	19	CS
…	…	…	…	…

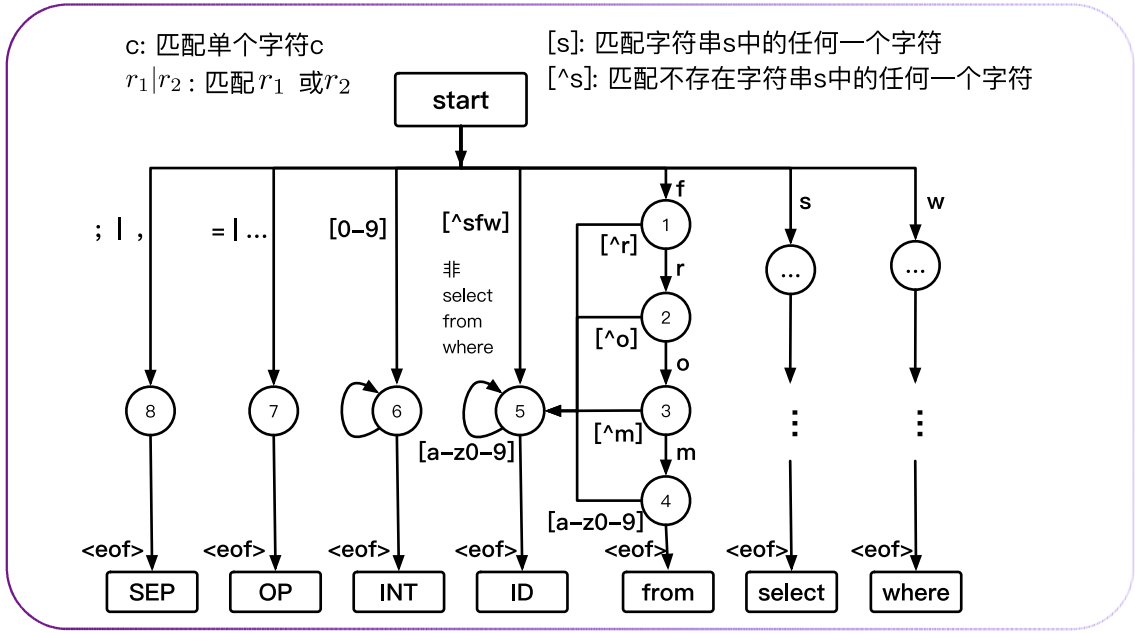
SC关系

Sno (学号)	Cno (课程号)	Grade (成绩)
2021310721	5	98
2021310722	1	87
…	…	…

词法分析

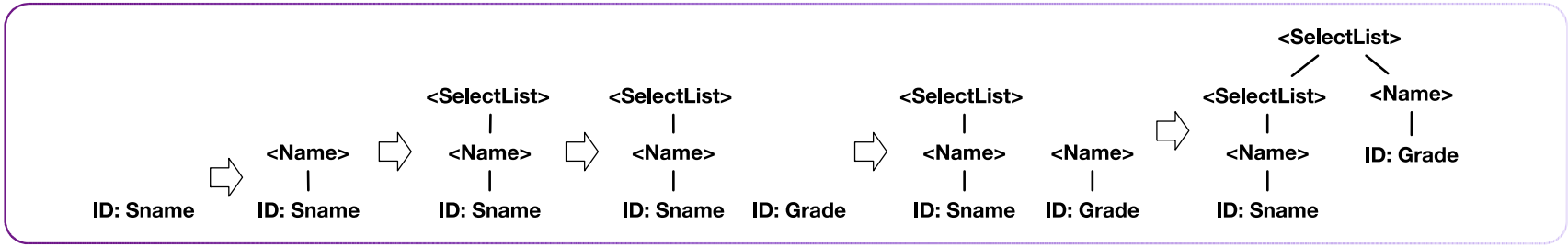
- SQL解析的第一步是对SQL查询进行词法分析
- 按照查询语句的词法规则将查询语句拆解为单词（token）序列，并提取关键字、标识、常量等
- 对不同类别的单词进行类别标记，在此基础上形成字符标记流

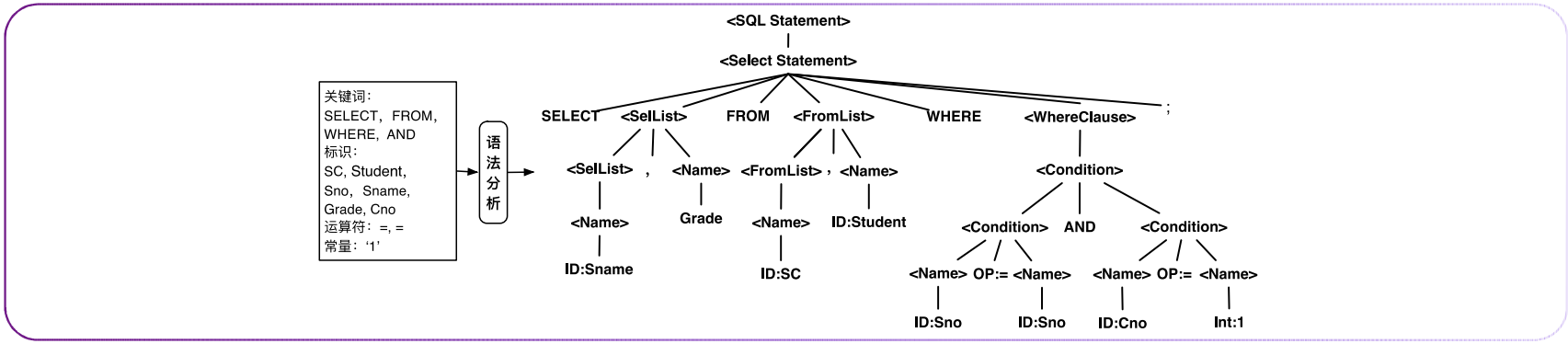




语法分析

- **根据词法分析输出的原子节点构建一颗语法分析树**
- 首先按照查询语言的语法规则判断SQL语句是否符合语法限制
- 如果SQL查询符合语法规则，则按照语法规则识别出SQL查询中的语法类节点，并结合词法分析输出的原子节点构建出一棵语法分析树





语法分析树

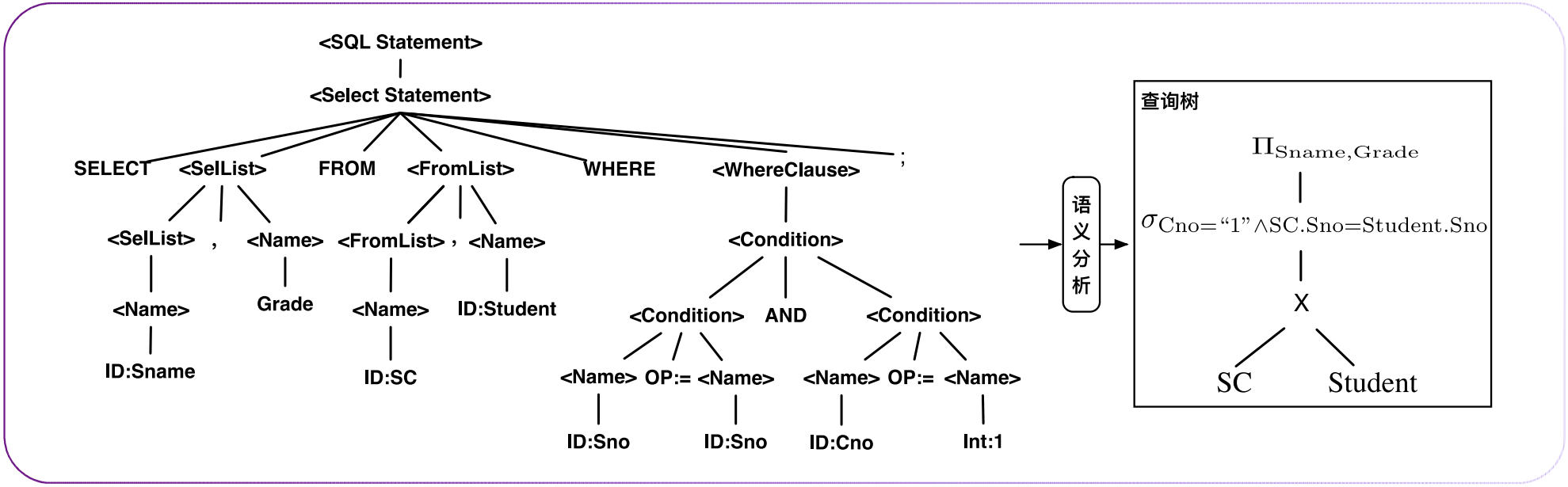
- **由原子节点和语法类节点构成，是语法分析的输出结果**
- **原子节点：**指词法分析过程后被标记的单词；来源于词法分析，在语法分析树中作为叶子节点出现，不存在后继节点
- **语法类节点：**使用<名称>的形式来表示。
  - 如< Select Statement >表示SELECT-FROM-WHERE形式的查询，
  - <FromList>表示跟在FROM之后的关系列表，
  - <Condition>表示作为筛选条件的表达式，常常用于表示在WHERE后的限制，
  - <SelectList>表示在SELECT后的属性列表，
  - <Name>表示属性名或关系名，
  - <Value>表示筛选条件中出现的常量值

语义分析

- **语义分析要对语法分析所输出的语法分析树进行语义上的正确性检查，并将语法分析树转化为方便查询优化阶段处理的查询树**
- **首先对语法分析树进行下述检查：**
  - **对关系（表名）的检查**：检查查询中出现的关系是否在关系模式中能找到匹配的关系或视图
  - **对属性（列名）的检查**：检查查询中涉及的属性是否能在数据库关系中找到匹配项
  - **对数据类型的检查**：检查查询中的运算是否与涉及的属性类型相匹配

语义分析

- 除了对语法分析树进行语义上的正确性检查外
- 语义分析还会对语法分析树做逻辑等价转化，将其转化为基于关系代数式表达的查询树便于后续的查询优化



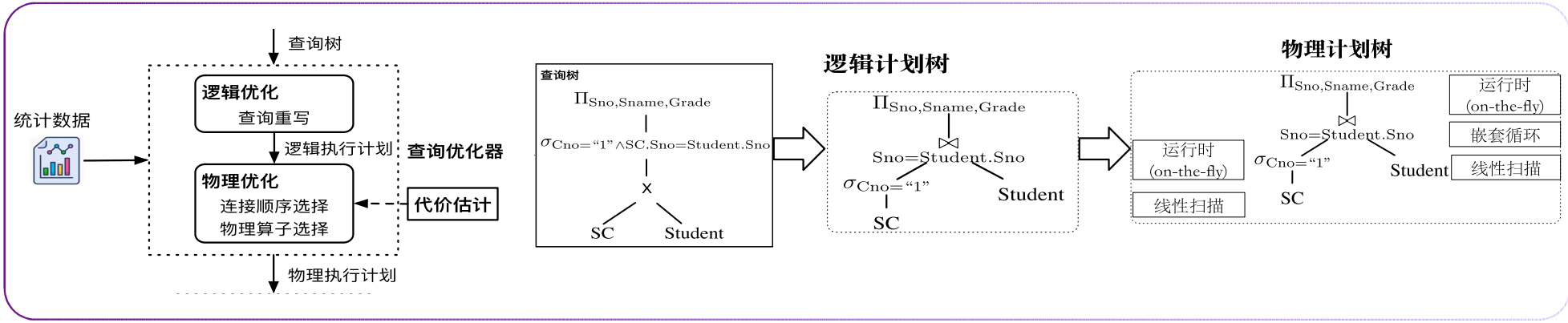
目录

查询优化概述

- **查询优化的动机：**
  - 同一条查询可以转换为多个逻辑等价的执行计划
  - 首先这些不同的逻辑执行计划在执行效率上存在差异
  - 其次同一个逻辑算子在数据库中有多种不同的物理实现方式，而不同场景下不同的物理实现方式的效率也存在差异
- **为了保证查询能以较低的资源消耗执行，数据库在查询解析和查询执行这两个过程之间存在查询优化的过程**

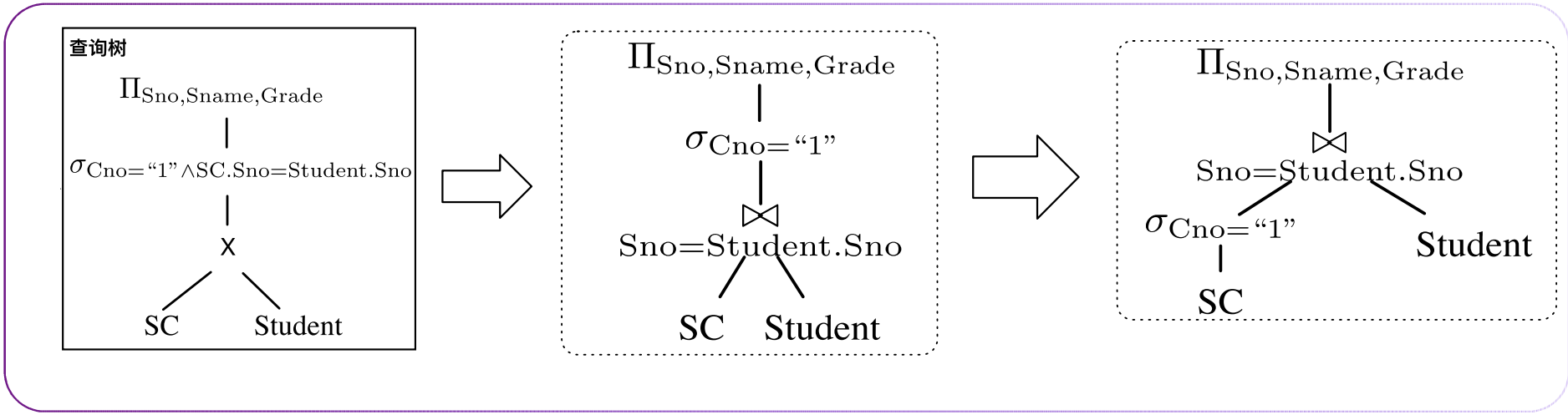
查询优化概述

- **查询优化的输入为查询树，对查询树依次从逻辑和物理两个层面进行优化，最终输出是查询的物理执行计划**
- **查询优化主要有三个重要的过程：**
  - 逻辑优化
  - 代价估计
  - 物理优化



逻辑优化

- **逻辑优化的输入是查询树，输出是经过逻辑优化后的查询树**
- **逻辑优化是找出与查询等价但执行效率更高的关系代数表达式**
  - 这一过程的核心思想是：对查询树做逻辑等价变化以获得更高效的逻辑执行计划。
  - 实现逻辑优化的方法主要是查询重写，根据特定的重写规则对查询树做逻辑等价变化。



代价估计

- **经过逻辑优化过程，可以得到基于等价规则重写过的逻辑执行计划**
- **将逻辑计划转换为实际执行的物理计划的过程中，还需确定每个运算的具体实现方式，比如**
  - 两表连接时应该具体采用何种连接算法
  - 多表连接时还需要确定表之间的连接顺序
- **为了确定以何种物理计划执行查询消耗的磁盘和CPU资源最少，就需要代价估计模型来估计不同物理执行计划的资源消耗**

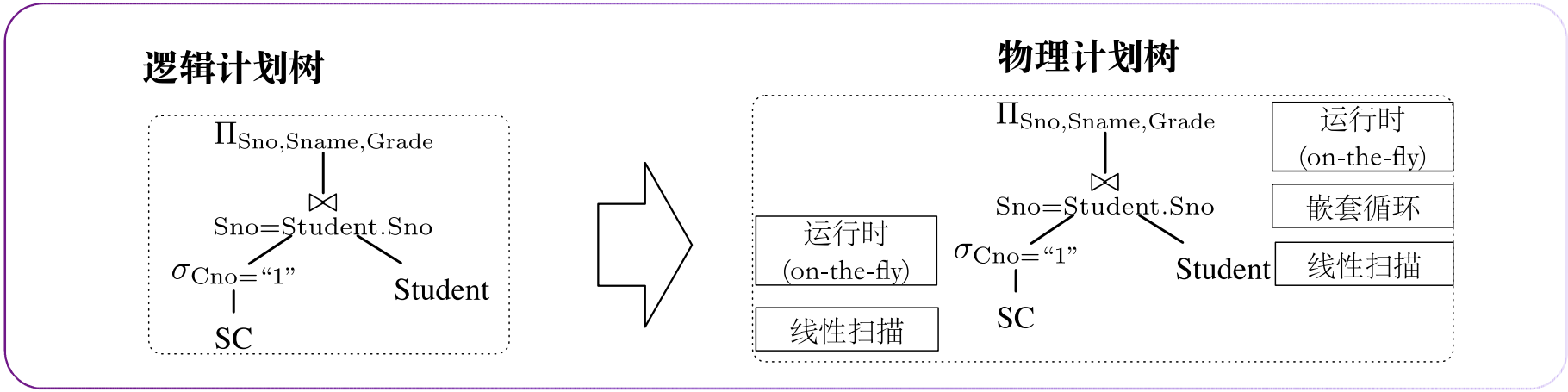
Sno (学号)	Cno (课程号)	Grade (成绩)
2021310721	5	98
2021310722	1	87
…	…	…

估计基数（结果）大小

2<Cno<4 ⇒ 30

物理优化

- **物理层面的优化则是枚举各种物理执行计划，根据代价估计模型择出代价最小的物理执行计划**
- **枚举不同的物理执行计划主要分为两方面：**
  - 枚举查询树中**逻辑算子的物理执行方式**，如连接算子是选择嵌套循环连接还是哈希连接
  - 当逻辑计划中出现**多表连接时枚举不同的连接顺序**
- **从中选择出代价最小的物理执行计划提供给查询执行引擎执行**



目录

查询算子概述

- SQL查询的执行过程，就像工厂的加工流水线，加工过程中的每一种工序都对应一种运算
- 这些运算可以被抽象为关系代数运算，查询算子是指这些关系代数运算
- 每种查询算子可能存在不同的物理实现方式
- 不同实现方式适合不同的场景

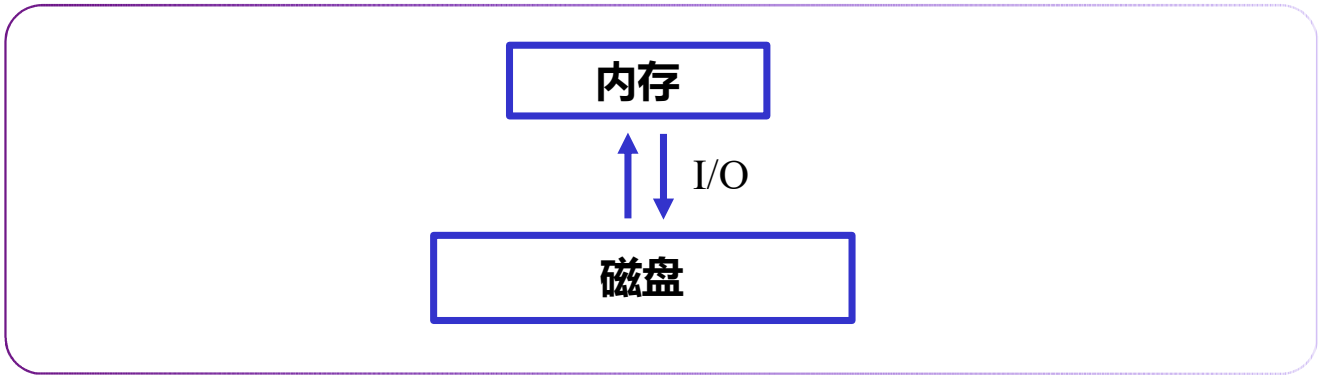
查询算子

- **各类查询算子及其实现方式汇总如下：**

查询算子	实现方式
排序算子	内存排序、外部归并排序
选择算子	线性扫描、索引扫描
连接算子	嵌套循环连接、 块嵌套循环连接、 索引嵌套循环连接、 排序归并连接 哈希连接
去重算子、聚集算子、集合算子	排序、哈希

磁盘I/O代价度量

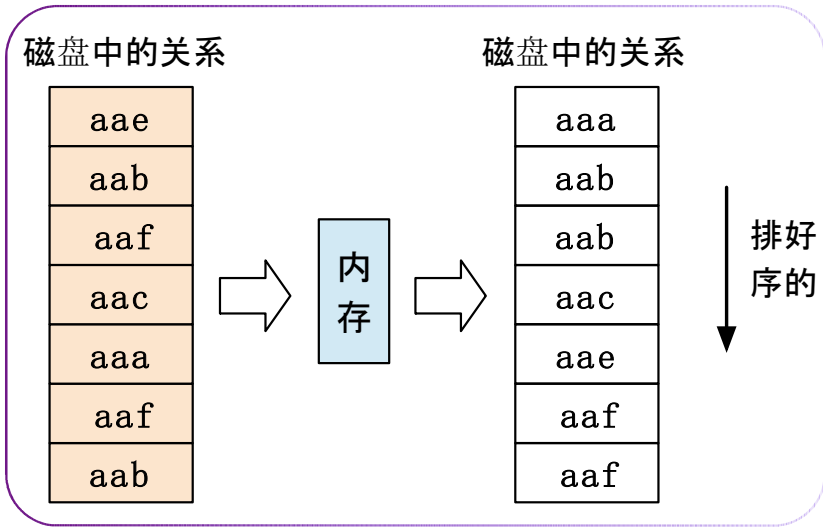
- 查询处理的代价主要是指查询对各种计算资源的消耗，包括磁盘I/O占用、执行查询所用的CPU时间，如果是分布式数据库的话还需要考虑数据通信代价
- 在大型数据库系统中，磁盘I/O代价是最主要的代价，可以**使用磁盘I/O的块数作为代价度量的指标**
- 为统一符号，使用**B(R)**表示关系R所占用的磁盘块数，**T(R)**表示关系R包含的元组条数，**M**表示可用的内存缓冲块数



目录

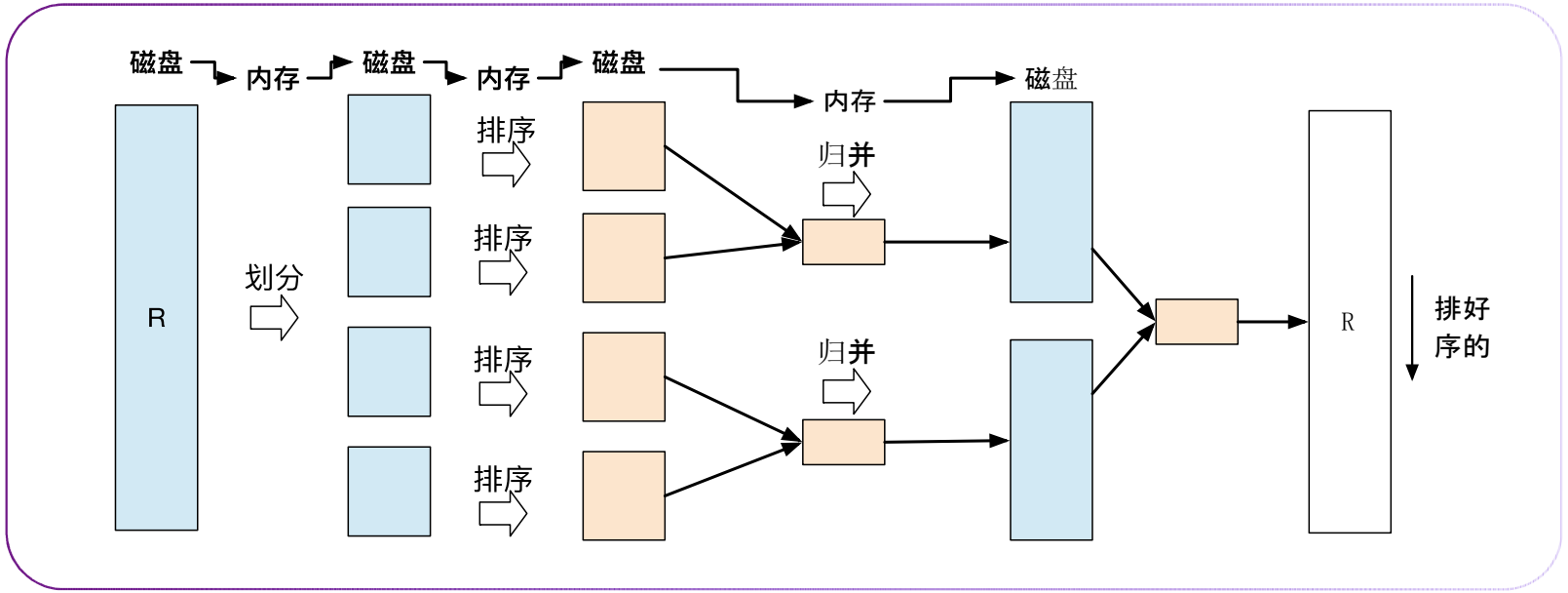
排序算子的实现与代价

- 当SQL查询中存在ORDER BY语句，该语句要求对结果中的元组进行排序
- 排序也是连接运算和其他运算（例如集合运算、去重运算）的关键步骤
- 主要考虑内存无法完全容纳需要排序的关系的情况
- 主要的实现方式是外部归并排序



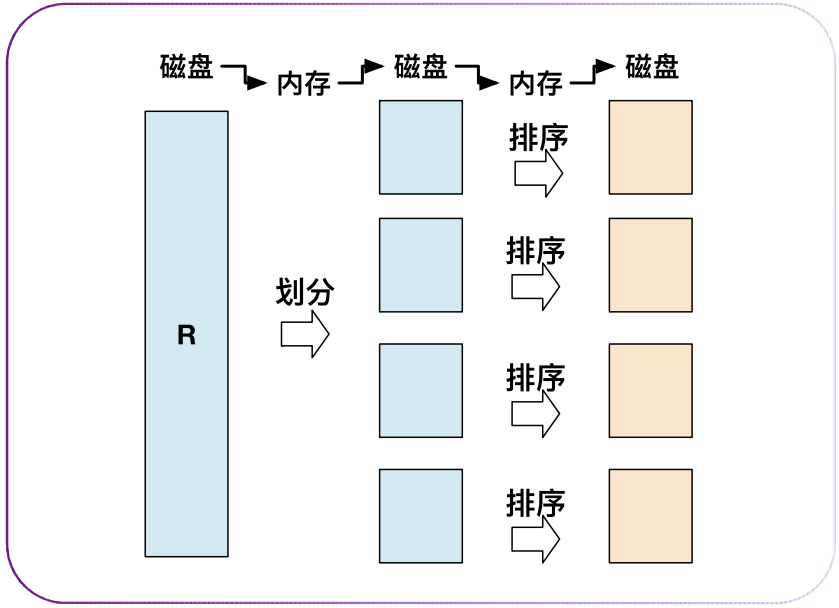
外部归并排序的实现

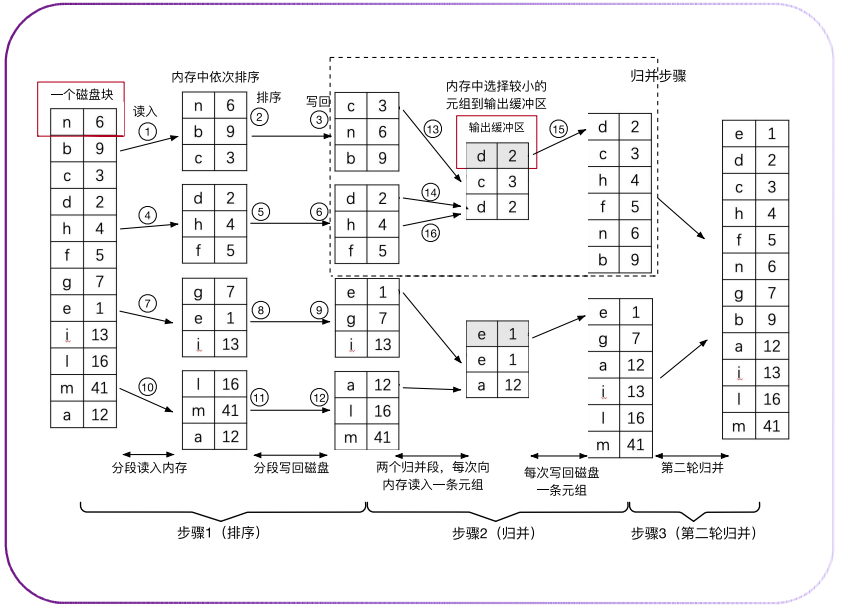
- 外部归并排序算法由两个阶段组成：**排序阶段和归并阶段**
  - **排序阶段**：首先划分多个归并段（run）
  - **归并阶段**：将已排序的归并段归并



外部归并排序的实现：排序阶段

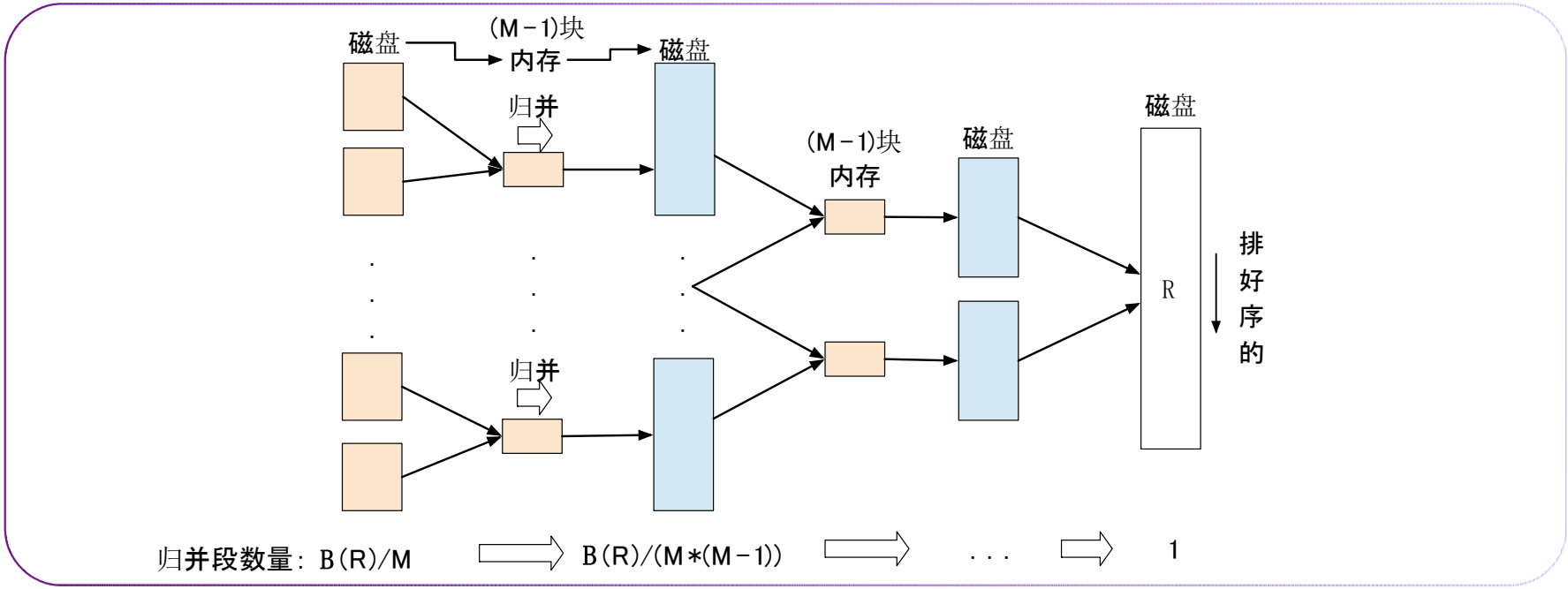
- **排序阶段：**首先划分多个归并段（run），各个归并段中包含的是需要排序的关系的部分元组。 其次将各归并段依次读入主存，在内存中排序并将排好序的归并段写回磁盘。
- 假设内存中可用的块数为M，因此初始归并段数为 $\lceil B(R) / M \rceil$





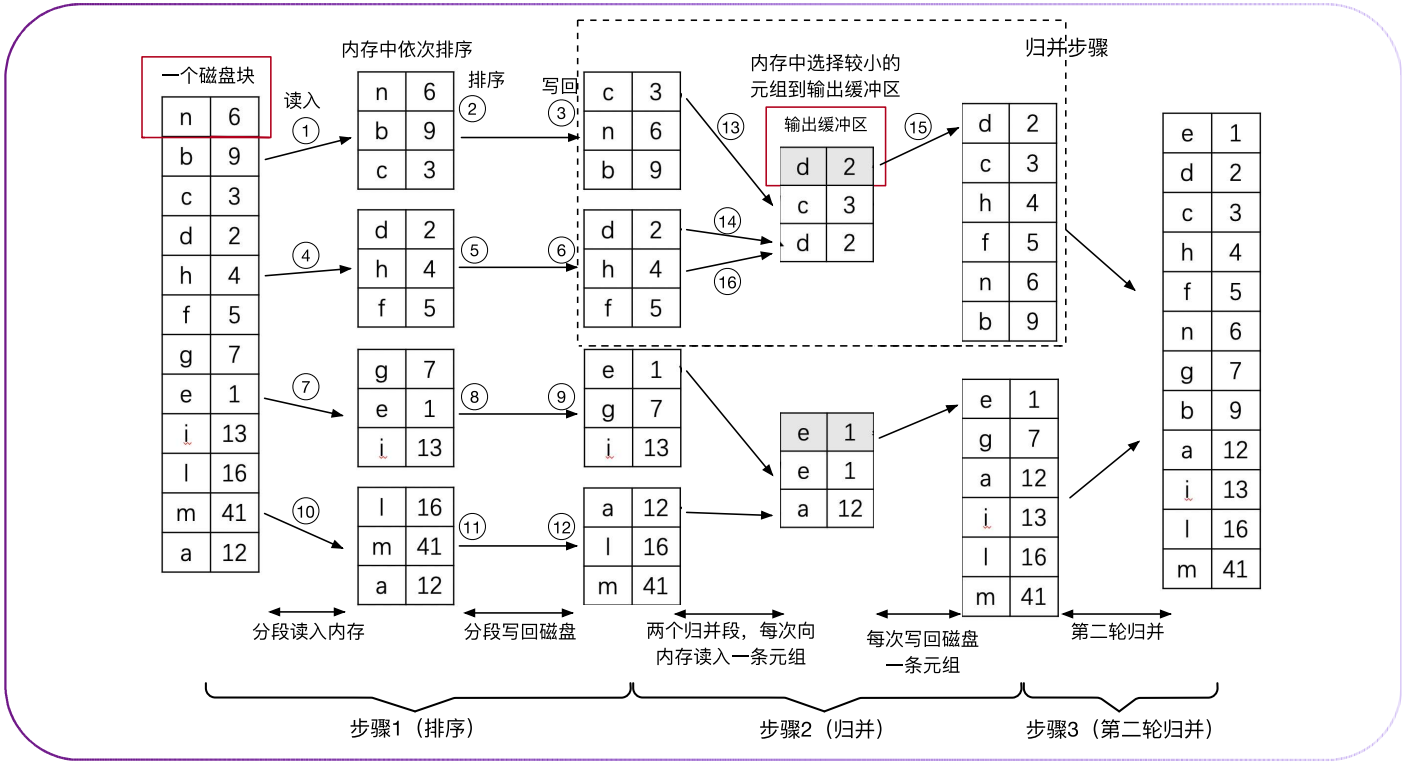
外部归并排序的实现：归并阶段

- **归并阶段**：已排序的归并段在一个或多个归并过程中被归并，每个归并过程可以有一个或多个归并步骤
- 初始归并段数为 $\lceil B(R) / M \rceil$，每一次归并过程使归并段的数量减少到原来的 $1/(M-1)$



外部归并排序例子—归并阶段

- 初始归并段数为 $[\mathrm{B}(R)\ /\ M]$，每一次归并过程使归并段的数量减少到原来的 $1/(M-1)$，因此归并过程数为 $[\log_{M-1}(\mathrm{B}(R)\ /\ M)]$



外部归并排序代价

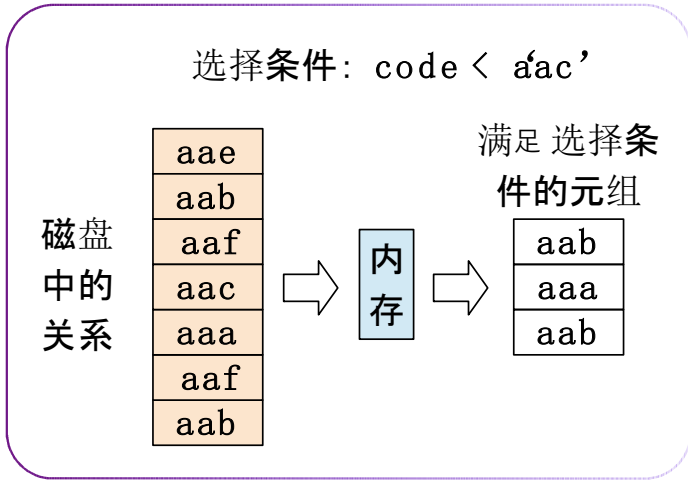
- 代价按照**排序和归并过程分为两部分：**
  - **1)** 排序阶段需要将关系 R 的所有元组读入内存并写回磁盘，因此排序的总磁盘I/O代价为$2\mathrm{B}(R)$；
  - **2)** 在归并阶段同样需要将所有元组读入内存并写回磁盘。因此每次归并阶段所需的磁盘I/O次数为$2B(R)$
    - 共需要 $\lceil \log_{M-1}(\mathrm{B}(R) / M) \rceil$ 次归并过程
- 因此外部归并排序的磁盘I/O代价为
  - $\boldsymbol{2(\lceil log_{M-1}(\mathrm{B}(R)/M) \rceil + 1) \cdot \mathrm{B}(R)}$

目录

选择算子的实现与代价

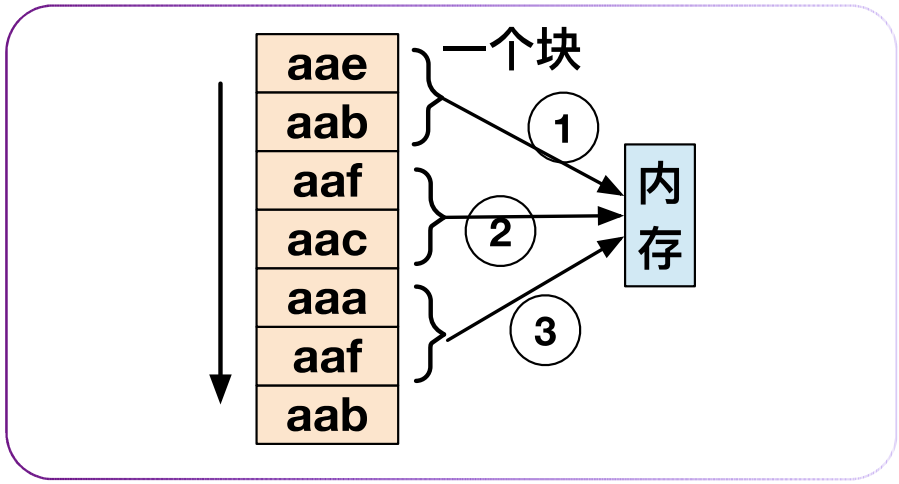
- 选择运算是查询处理中最常见的运算，实现选择运算最常见的算法是扫描整个关系并选出目标元组
- 通常可以根据关系上有无索引将关系扫描分为**线性扫描与索引扫描**两大类
- 不同场景下（例如：属性列上的索引是否为聚簇索引）的选择运算，实现的具体方式有所差异

扫描算法	应用条件
线性扫描	等值选择，范围选择
索引扫描	B^+ 树聚簇索引， B^+ 树非聚簇索引 等值选择，范围选择



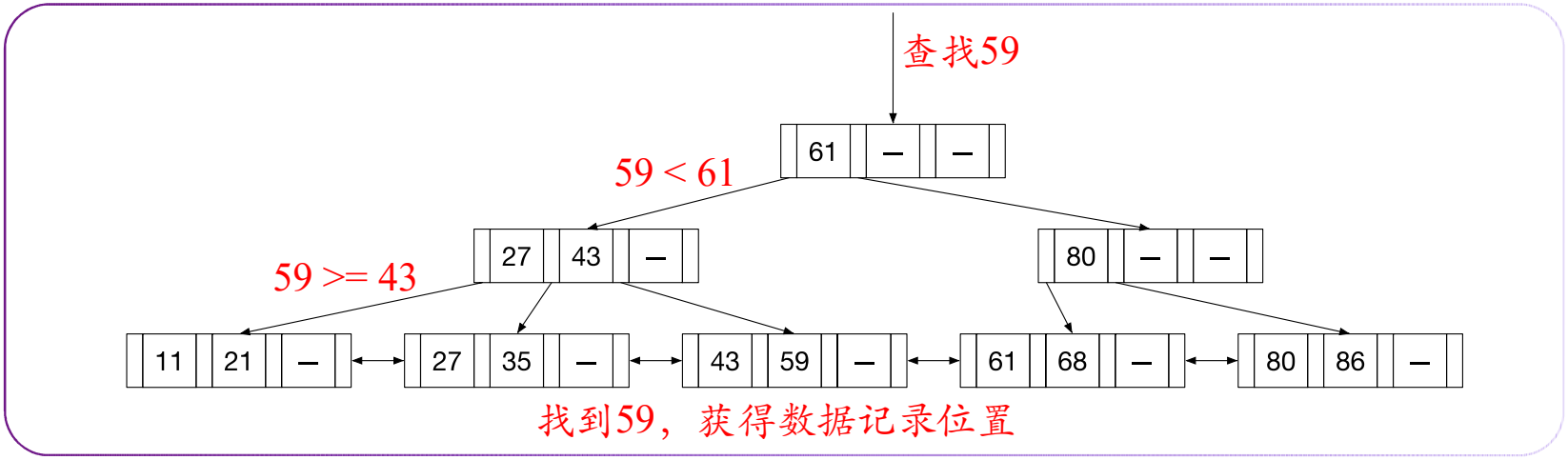
线性扫描

- 线性扫描开始时，需要进行一次磁盘搜索以使磁盘头对齐关系R在磁盘中占用的第一个块的物理位置。
- 之后依次读取关系R对应的每一个磁盘块到内存中，对读入内存中的元组进行检索，直至检索出满足选择条件的元组。
- 线性扫描可以在任何情况下实现选择运算，但线性扫描的磁盘I/O代价通常高于索引扫描。



索引扫描

- 当关系R中目标属性上存在索引时，可以利用索引执行扫描算法，这种算法称为索引扫描（index scan）。
- 本节讨论的索引主要是指B+树索引，记h_t为B+树的深度。
- 可以在该索引上使用第9章介绍的B+树查找算法找出满足选择条件的元组。

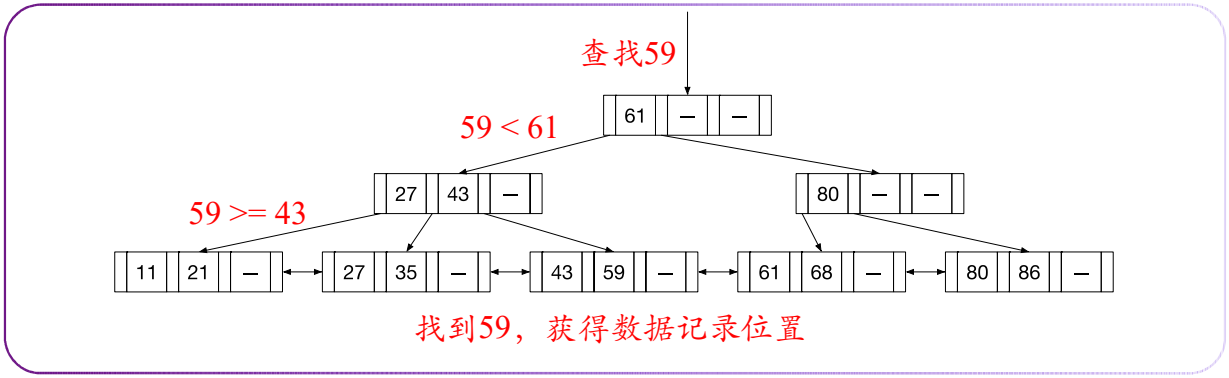


无索引下的等值选择

- **实现：**
  - 进行一次磁盘搜索以确定关系R对应的磁盘中的第一个块
  - 之后依次读取关系R对应的每一个磁盘块到内存中，在内存中顺序判断每个块中的所有元组是否满足等值选择条件
  - 直至扫描完关系R的最后一条元组
- **代价：**
  - 线性扫描的磁盘块数为 B(R)，因此磁盘I/O代价为 B(R)。
  - 当线性扫描作用的列上不存在重复值时，搜索块数的期望为 B(R)/2 ，此时磁盘I/O代价为 B(R)/2。

聚簇索引下的等值选择

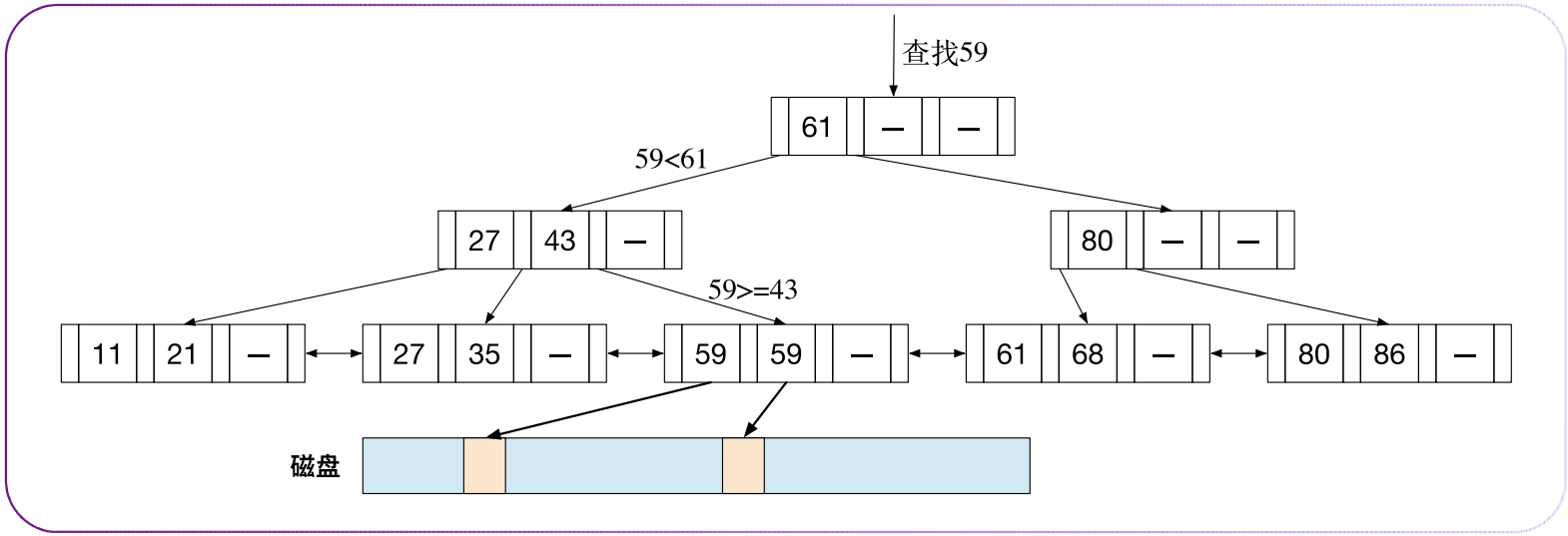
- **实现：**
  - 在B+树索引上使用B+树查找算法搜索到唯一符合条件的元组在磁盘中的位置
  - 再将该元组所在的磁盘块读入内存中，在内存中找出目标元组
- **代价：**
  - 由于B+树索引的结构，从磁盘中读取索引的次数为h_t，此外还需要执行一次磁盘读取以从磁盘中读取目标元组所在的磁盘块。
  - 每次磁盘读取需要一次完整I/O操作，则代价可以表示为h_t+1
  - h_t表示B+树索引的高度



非聚簇索引下的等值选择

➢ **实现：**

- 使用B+树查找算法搜索到多个符合条件的元组在磁盘中的位置
- 再将这些元组所在的磁盘块依次读入内存中，在内存中搜索出目标元组



非聚簇索引下的等值选择

➢ **代价**:

- 符合选择条件的元组可能有多个（记为n个）
- 辅助索引查找键顺序与数据文件记录顺序不一致，也就是符合选择条件的每条元组的存储位置可能位于物理位置不相邻的磁盘块
- 与聚簇索引下的等值选择的区别是每条符合选择条件的元组都需要一次磁盘I/O
- 则该情况下的选择运算总代价是$\mathrm{h}_t + n$

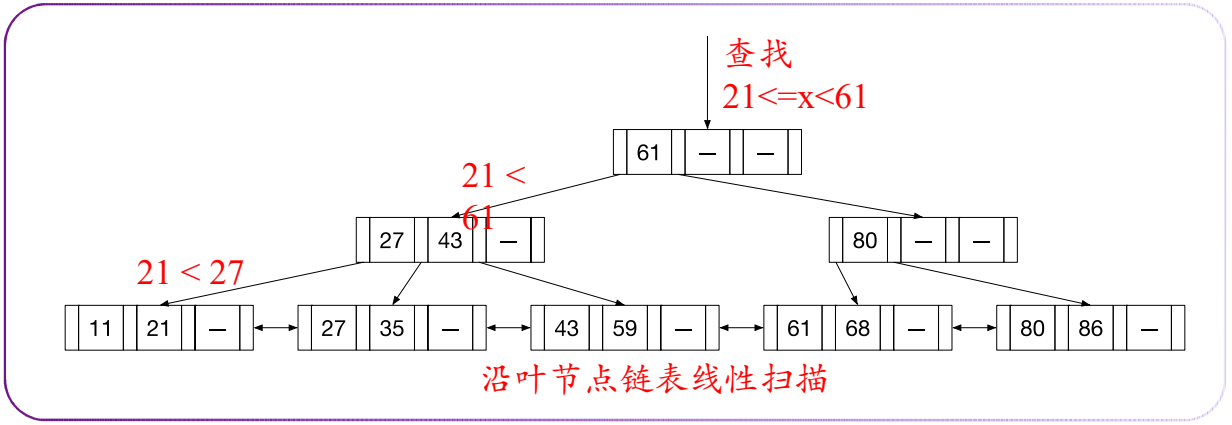
无索引下的范围选择

- **实现**
  - 使用线性扫描，与无索引下的等值选择的算法实现相同，区别是筛选元组的选择条件不同
- **代价：**
  - 该运算的磁盘搜索块数为 B(R)，因此磁盘I/O代价为 B(R)

聚簇索引下的范围选择

➢ 实现

- 在建有聚簇索引的主属性上执行范围选择时，可以在B+树索引上使用B+树查找算法搜索出运算符为等值运算时的第一条元组
- 如果运算符为 “<”，则选择关系R中在此条元组之前的所有元组；若为 "≤"，则选择R中此条元组及其之前的所有元组
- 当运算符为 ">","≥" 时同理



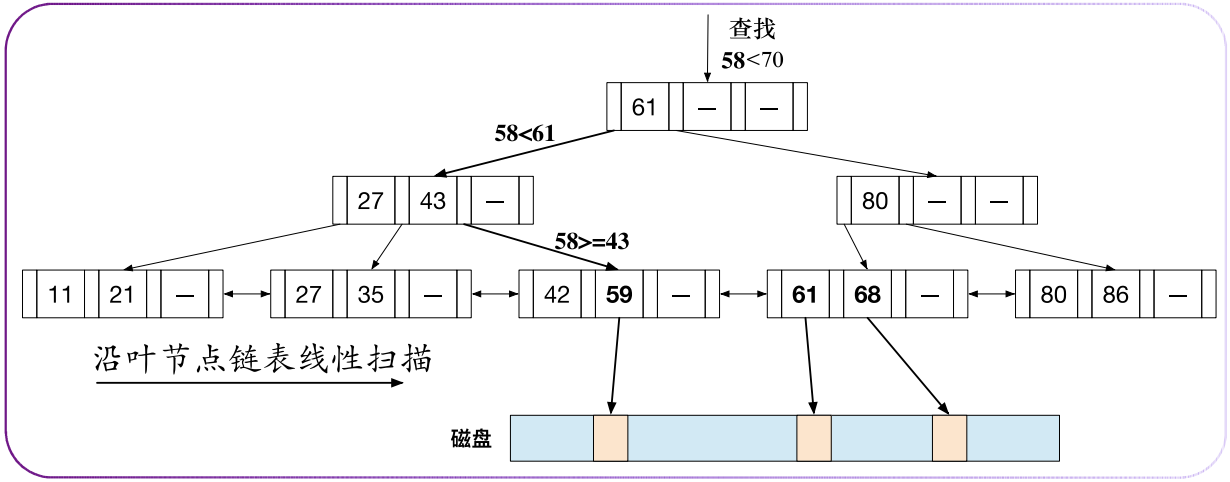
➢ 代价

- 记符合选择条件的元组**对应的磁盘块数**为b
- 将b个磁盘块分多次读入内存，则该情况下的选择运算总代价是$h_t + b$

非聚簇索引下的范围选择

➢ 实现

- 与非聚簇索引下的等值选择的区别在于等值选择只需搜索B+树中满足等值条件的叶子节点
- 而范围选择需要搜索满足等值条件的B+树叶子节点之前或之后的所有叶子节点



➢ 代价

- 与非聚簇索引下的等值选择的代价分析过程相同
- 假设符合范围选择条件的元组有n个，则其磁盘I/O代价为h_t + n

选择运算代价汇总

扫描算法	选择运算应用条件	运算代价	分析
线性扫描	等值选择	B(R)	B(R) 个块的读写，平均代价为B(R)/2
索引扫描	B^+ 树聚簇索引，等值	$h_t + 1$	h_t次读取索引，读取数据一次（h_t索引高度）
索引扫描	B^+ 树非聚簇索引，等值	$h_t + \mathrm{n}$	h_t次读取索引，读取n个满足条件的条数（不连续）
线性扫描	范围选择	B(R)	B(R) 个块的读写
索引扫描	B^+ 树聚簇索引，范围	$h_t + \mathrm{b}$	h_t次读取索引，读取b个满足条件的块数(连续)
索引扫描	B^+ 树非聚簇索引，范围	$h_t + n$	h_t次读取索引，读取n个满足条件的条数（不连续）

合取选择

- **即多个选择条件用关键词“AND”连接的情况**
- **无索引：**
  - 线性扫描
- **单列索引：**
  - 根据该索引B+树查找算法检索出满足单个选择条件的所有元组
  - 检查每个检索到的元组是否满足剩余的选择条件
- **多个单列索引：**
  - 找出基数估计值最小的选择条件对应的列，根据索引检索出满足该选择条件的所有元组
  - 检查检索到的元组是否满足剩余选择条件；或与其他列索引检索出的元组求交集
- **复合键索引：**
  - 如果合取选择作用的多个列上建立有复合键索引
  - 可以通过查找该复合键索引直接检索出满足选择条件的元组

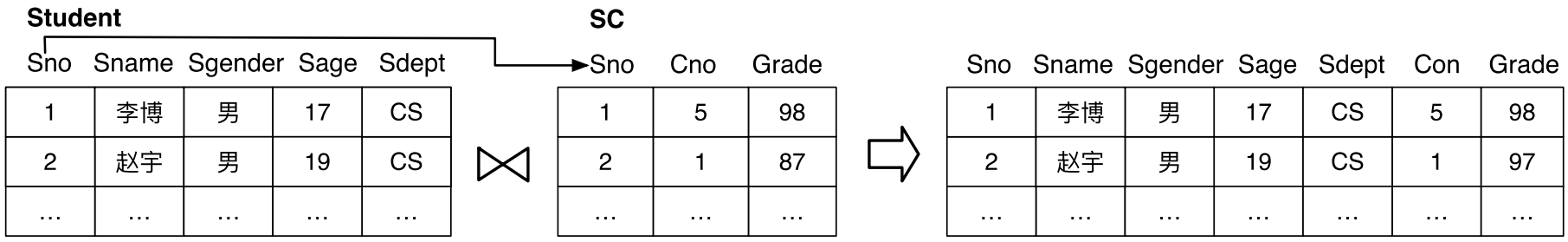
析取选择

- **即多个选择条件用关键词“OR”连接的情况**
- **无索引：**
  - 线性扫描
- **所有要选择的列上都有索引：**
  - 通过B+树查找算法分别检索出满足每个选择条件对应的元组集合，然后对这些集合应用集合运算中的并操作来获得析取选择的结果

目录

连接算子的实现与代价

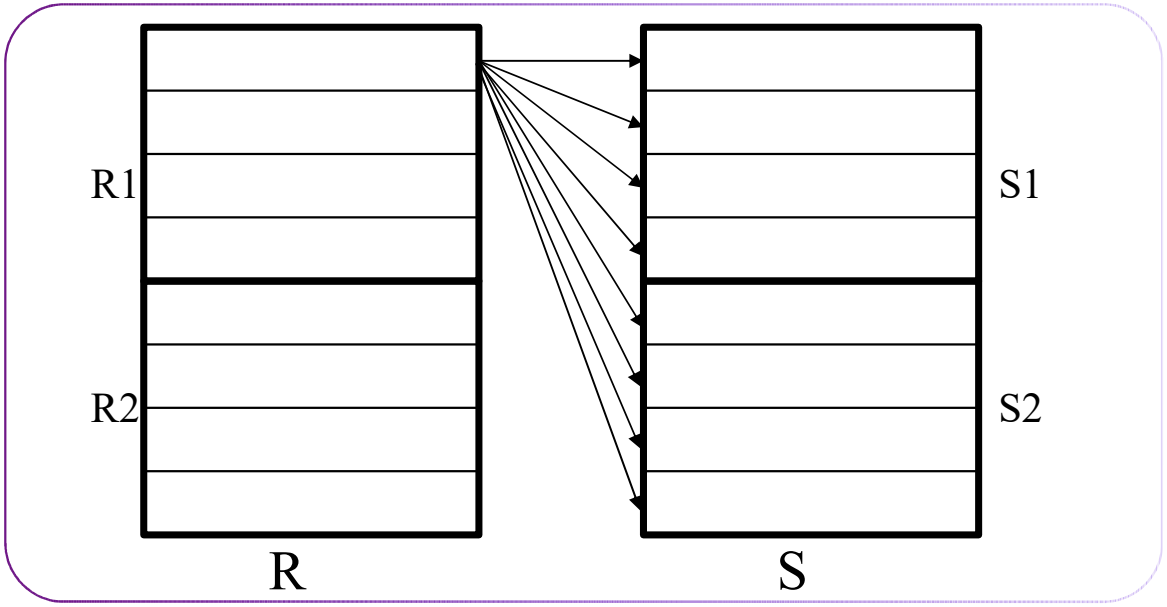
- **关系连接是数据库中最常见且最耗时的运算之一**
  - 如图，展示了两个简化的SC和Student表的等值连接过程
  - 两个表进行连接的一种逻辑等价的描述为：枚举两个表中所有行的组合，即两个表的笛卡尔积，然后返回满足连接条件（图中是等值连接条件）的组合



嵌套循环连接

- **除了关系R，S分块读入内存中的循环外，该算法由两个嵌套的循环过程构成，因此称它为嵌套循环连接算法**
- **嵌套循环连接**
  - 在外层循环中的关系R被称为外层关系，而S被称为内层关系
  - 外层每次读入R的一条记录
  - 内存每次读入S的一条记录
- **嵌套循环连接R⋈ S算法：**

```
for r in R
   for s in S
      if (r,s) is true
         add(res, (r, s))
```



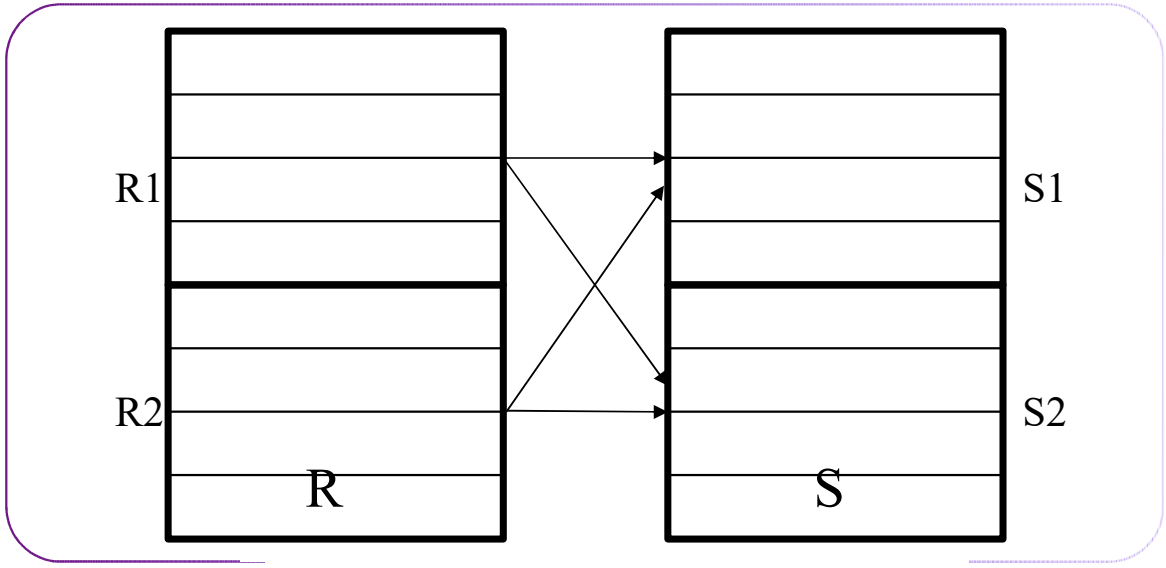
嵌套循环连接

- **代价分析：**
  - 由于关系R是分块读入内存，因此对于外层关系R的磁盘I/O次数为B(R)
  - 最外层循环是对关系R中的每个记录迭代，则外层循环的迭代次数为T(R)
  - 对于每次外层循环的迭代，需要按块将关系S读入内存，读入的内存块数为B(S)，
  - 对于内层关系S的磁盘I/O次数为T(R)· B(S)
- **嵌套循环连接的磁盘I/O总代价为B(R)+T(R)· B(S)**

块嵌套循环连接

- **嵌套循环连接中，对于外层循环的每次迭代只将一个记录读入内存,导致外层循环的迭代次数过多，造成对内层关系S需要更多次的读取**
- **块嵌套循环连接**
  - 对于外层关系R则每次读入1块到内存中
  - 对于内层关系S则每次读入1块到内存中
- **块嵌套循环连接：**

```
for BR in R
   for BS in S
     for r in BR
       for s in BS
         if (r,s) is true
            add(res, (r,s))
```



块嵌套循环连接

- **代价分析：**
  - 由于关系R是分块读入内存，因此对于外层关系R的磁盘I/O次数为B(R)
  - 最外层循环是对关系R中的每个块迭代，则外层循环的迭代次数为B(R)
  - 对于每次外层循环的迭代，需要按块将关系S读入内存，读入的内存块数为B(S)
  - 因此对于内层关系S的磁盘I/O次数为B(R)· B(S)
- **块嵌套循环连接的磁盘I/O总代价为B(R)+B(R)· B(S)**

优化的块嵌套循环连接

- 块嵌套循环连接对于外层关系R则每次读入M-1块到内存中
- 然后内层关系S读入一块到内存
- 块嵌套循环连接：

for BR_{M-1} in R

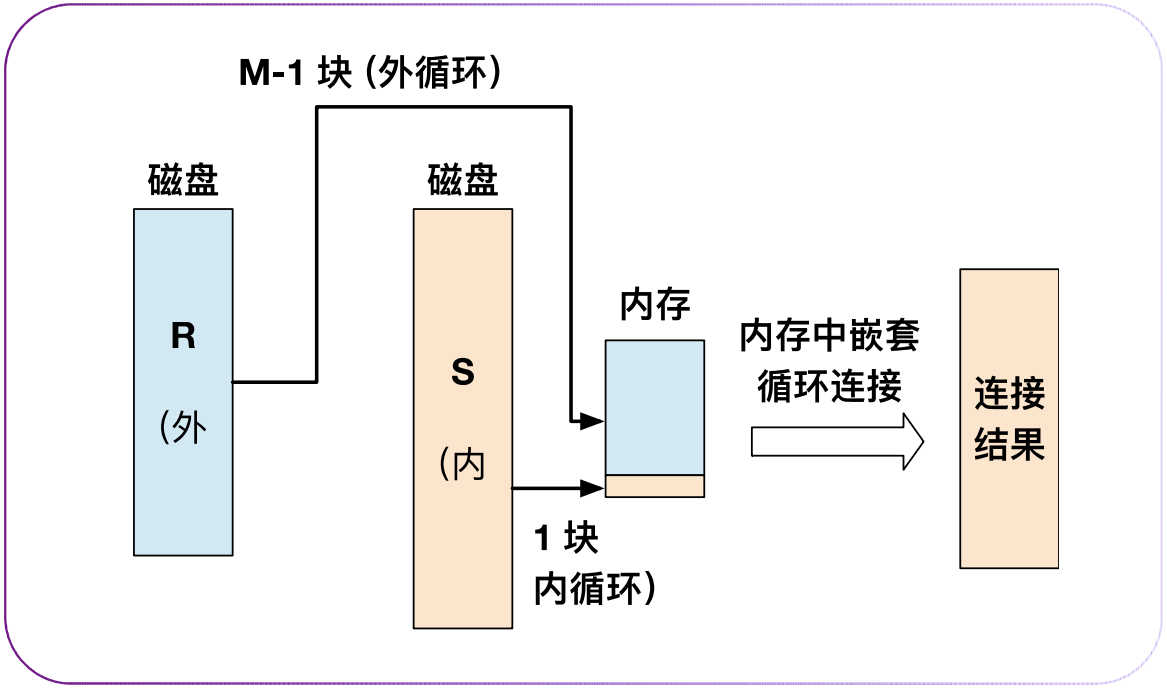
for BS_1 in S

for r in BR_{M-1}

for s in BS_1

if (r,s) is true

add(res, (r,s))

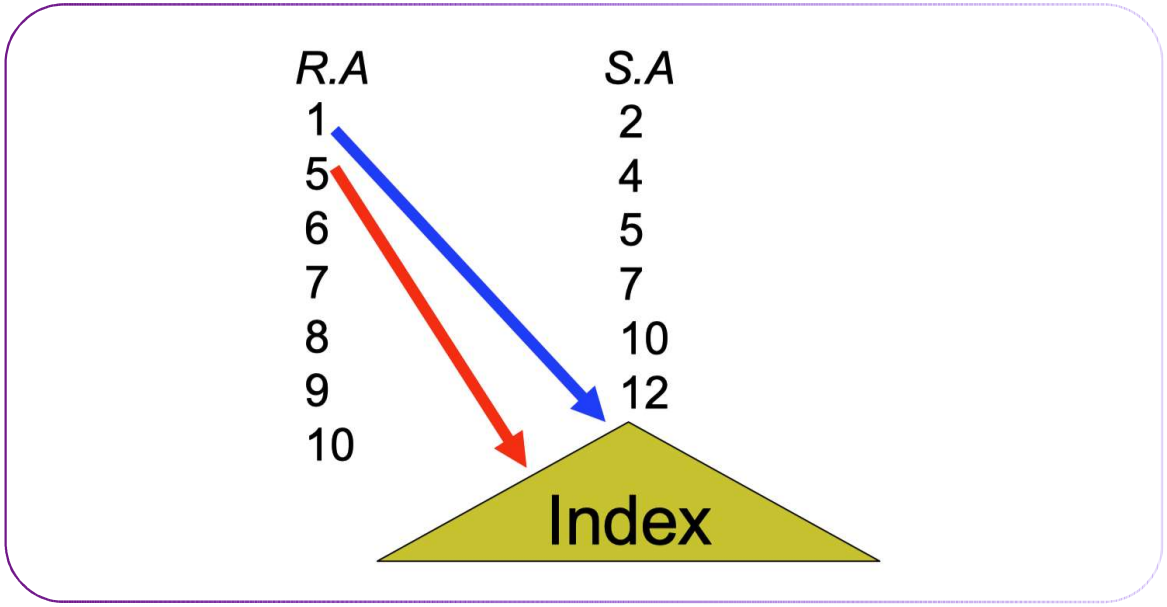


块嵌套循环连接

- **代价分析：**
  - 由于关系R是分块读入内存，因此对于外层关系R的磁盘I/O次数为B(R)
  - 外层循环将关系R每次读M-1个磁盘块到内存，最外层的迭代次数为B(R)/(M-1)
  - 对于最外层的每次迭代，会读取关系R的M-1个块和关系S的B(S)个块
  - 总的磁盘I/O代价为B(R)+(B(S)· B(R))/(M-1)
- **可以选择B(R),B(S)较小的作为外层**
- **因此总的磁盘I/O代价为 Min(B(R),B(S))+(B(S)· B(R))/(M-1)**

索引嵌套循环连接

- 当内层关系 S 的连接属性上存在索引，则可以将嵌套循环连接算法中的线性扫描用索引扫描来代替
- 即对于外层关系 R 中的每个元组r，可以利用关系S上的索引选择出满足连接条件的元组s并将（r,s）放入结果中



索引嵌套循环连接

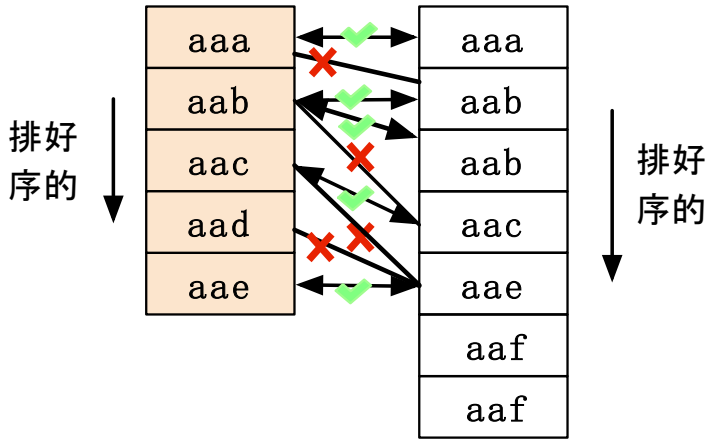
- **给定外层关系R的一条元组r的情况下，在关系S中查找满足连接条件的元组本质上是在内层关系S上做选择运算，而选择条件便是连接条件**
- **代价分析：**
  - 读取关系 R 需要B(R)次磁盘I/O
  - 对于关系 R 上的每个元组，在 S 中需要根据索引搜索出满足连接条件的元组，此部分的代价对应索引选择运算的实现代价
- **因此，计算代价的公式可以表示为 B(R)+T(R)•c**
  - 其中 c 指的是对于关系R的每条元组在关系 S 上根据连接条件进行选择的单次运算的平均代价

排序归并连接

- **嵌套循环连接算法需要对所有可能的元组对进行筛选，这带来了很高的代价。**
- **实现方式：**
  - 排序：将需要连接的两个关系根据连接属性进行排序
  - 连接：按照排序好的连接属性顺序分别扫描这两个关系，满足连接条件的元组将被组合成对以得到结果关系
    - 如果r=s，则将（r，s）添加加到连接结果中；
    - 如果r<s，则将r移除，并访问R的下一个元组；如果r>s，则将s移除，并访问S的下一个元组；
    - 如果r>s，则将s移除，并访问S的下一个元组；

排序归并连接

- **考虑在关系 R 、S 上，以 $R.A$、$R.B$ 这两个整数值属性等值连接的情况，该算法的伪代码:**



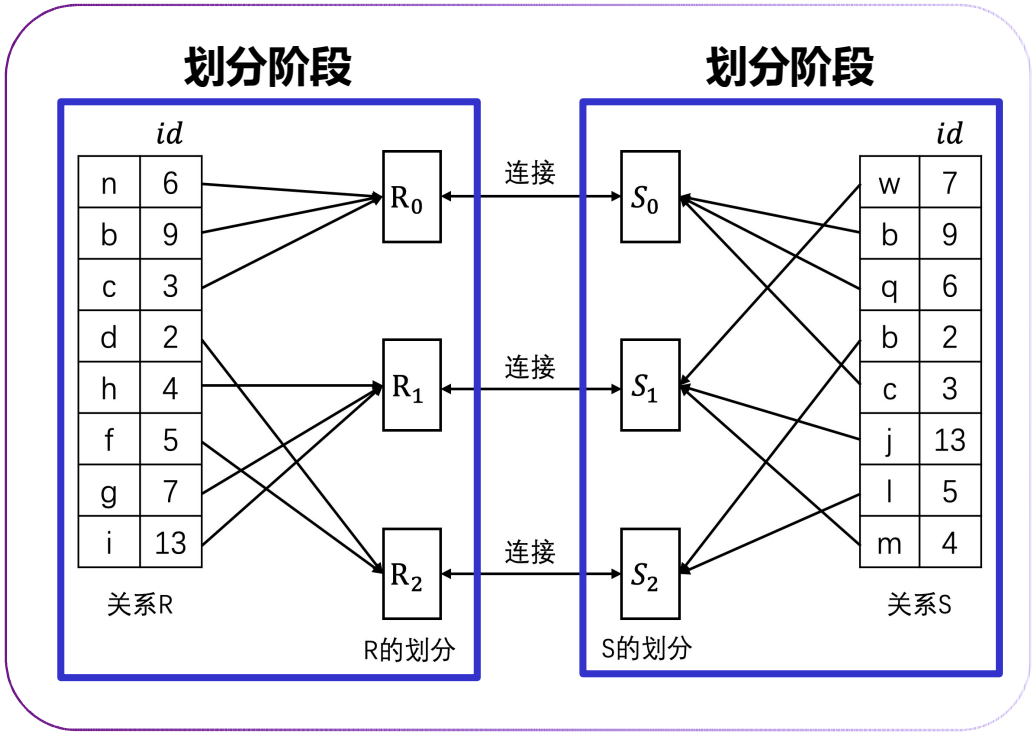

```
// 函数：排序归并连接的归并阶段
// 参数R,S：待归并连接的两个关系R,S
function merge_join(R,S)
        sort_r(R,R.A)  // 根据R.A 排序R
        sort_s(S,S.B)  // 根据S.B 排序S
        r = R 的第一个元组
        s = S 的第一个元组
        while r ≠ Null and s ≠ Null // 未遍历完两个关系时进入循环
                while r.A > s.B
                        s = next(S)          // S中的
                while r.A < s.B
                        r = next(R)          // R中的r的下一条元组

                // 如果R，S中没有重复元组，只需将(r,s)放入结果
                // 如果R，S中有重复元组，则需要找到所有满足条件的元组对
                while r.A == s.B
                        s' = s               // 记录S中当前满足条件的第一条元组
                        while r.A == s'.B    // 循环S中满足条件的重复元组
                                将(r,s') 放入结果
                                s' = next(S)  // S中的 s' 的下一条元组
                        r = next(R)          // R中的r 的下一条元组
                s = s'
        return res
```

排序归并连接

- **排序归并连接算法对每个关系中的元组只需扫描一次，即扫描 |R|+|S| 个元组**
- **相比嵌套循环连接需要扫描 |R|· |S| 个元组相比大大减少了磁盘访问次数**
- **但排序归并连接比嵌套循环连接多出了对两个关系进行排序的步骤**
- **代价分析：**
  - 假设两个关系R, S是已经排好序的关系，归并连接运算过程中只需要对两个关系顺序读取一次，因此磁盘I/O次数是 B(R)+B(S)

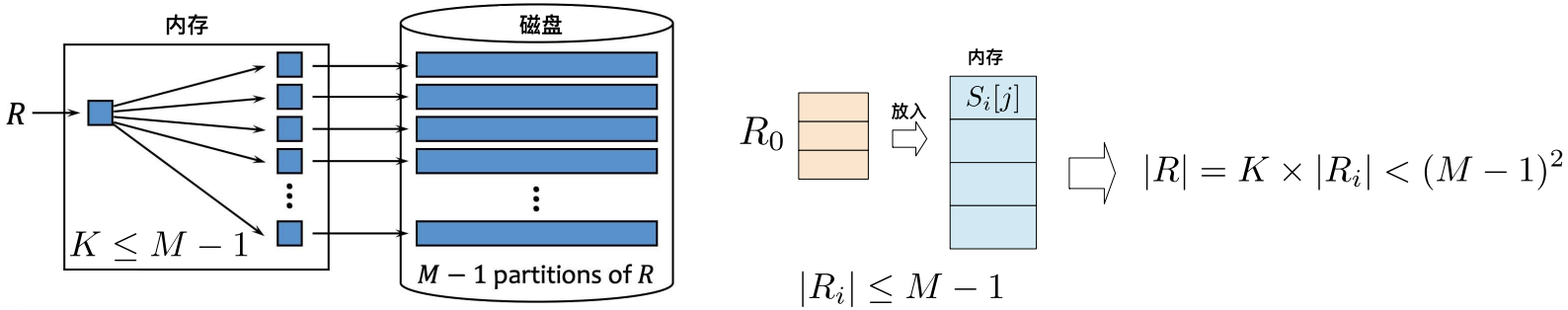
哈希连接

- **哈希连接分为两个阶段，划分阶段和探查阶段**
  - **划分阶段：**
    - 根据设置好的哈希函数h对关系R和S进行划分，
    - 关系R被划分为K个桶 $R_0, R_1, \cdots R_{K-1}$
    - 关系S被划分为K个桶 $S_0, S_1, \cdots S_{K-1}$。
    - 其中哈希函数 h 将连接属性值作为参数，将哈希值相同的元组划分到同一个桶
  - **探查阶段：**每两个对应的桶 R_i 和 S_i 被连接
- **探查阶段共需 K 次迭代。在第i次迭代期间，R_i 和 S_i 的连接使用其他连接算法**
  - 嵌套循环连接
  - 排序归并连接
  - 哈希连接



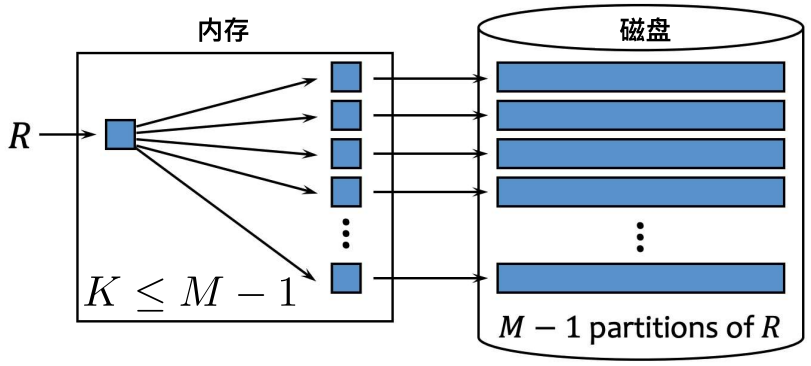
哈希连接代价

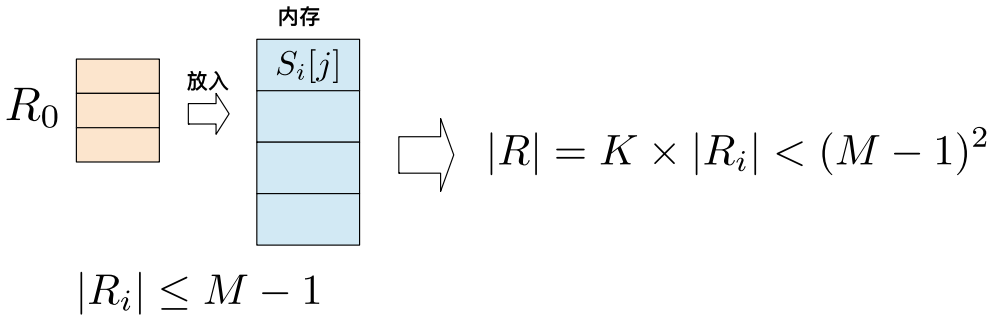
- **代价分析：**
  - 划分阶段的磁盘读取和写回，以及探查阶段的磁盘读取和写回
  - 在划分阶段对两个关系 R,S 各需要一次读取和写回：2(B(R)+B(S))
  - 探查阶段需要对每个关系划分读取一次，则需要B(R)+B(S)
- **哈希连接的磁盘I/O代价为：3(B(R)+B(S))**
- **约束：哈希桶数能放到内存，每个哈希桶也能放到内存**



哈希连接约束

- **约束条件：**上述的哈希连接算法对可用内存块的数量M有一个要求，即$\boldsymbol{M > \sqrt{\mathrm{B}(\boldsymbol{R})} + 1}$。
- **分析：**
  - 划分阶段所需的内存缓冲块的数量至多为 M-1。因此关系R 的任意一个划分 R_i 至少包含 $\frac{\mathrm{B}(R)}{M-1}$ 个磁盘块。
  - 在探查阶段，需要保证有足够的内存来存放关系S的一个磁盘块以及关系R的任意一个划分 R_{i}，因此就有$M-1>\frac{\mathrm{B}(R)}{\mathrm{M}-1}$，即$M>\sqrt{\mathrm{B}(R)}+1$。





连接运算代价汇总

连接算法	连接运算应用条件	连接运算代价
嵌套循环连接	适用所有情况	$\mathrm{B}(R)+\mathrm{T}(R)\cdot B(S)$
块嵌套循环连接	适用所有情况	$\mathrm{B}(\mathrm{R})+\big(B(S)\cdot B(R)\big)/(M-1)$
索引嵌套循环连接	当关系S上建有索引	$\mathrm{B}(\mathrm{R})+T(R)\cdot c$
排序归并连接	需要先将关系R和S排序	$\mathrm{B}(\mathrm{R})+\mathrm{B}(\mathrm{S})$
哈希连接	适用所有情况 $\mathrm{M}>\max(\sqrt{\mathrm{B}(R)},\sqrt{\mathrm{B}(S)})+1$	$3\ (\mathrm{B}(\mathrm{R})+\mathrm{B}(\mathrm{S}))$

目录

其他算子的实现与代价

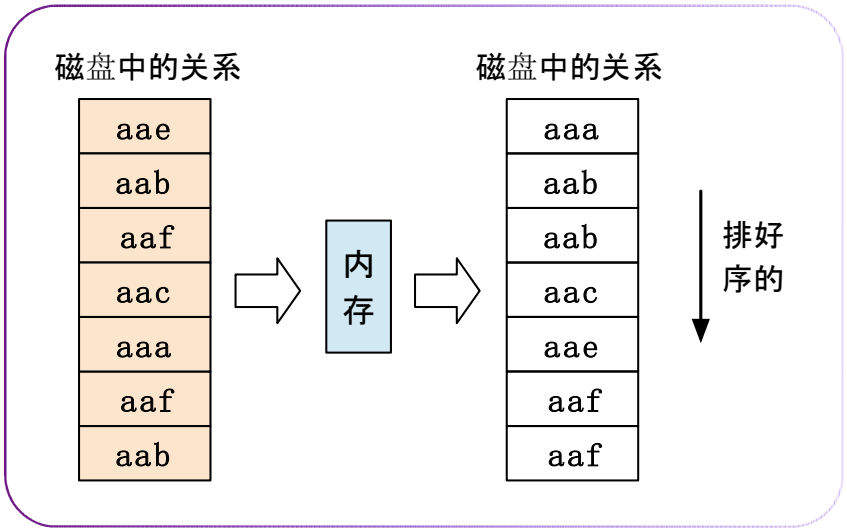
- **除了选择、排序、连接这些最主要的运算**
- **还有其他的运算需要考虑**
  - 去重运算
  - 集合运算
  - 分组聚集运算
- **这些运算都是基于排序和哈希划分来实现**

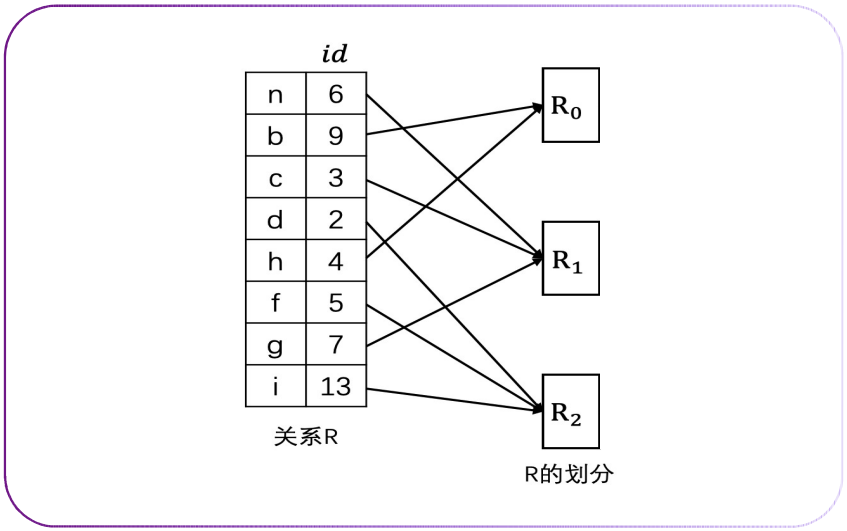
单个关系排序与哈希划分

- **排序的实现是指外部归并排序**
  - 根据外部归并排序算法对无序的关系R排序，其磁盘I/O代价为

    $2(\lceil \log_{M-1}(\mathrm{B}(R)/M)\rceil + 1)\cdot \mathrm{B}(R)$

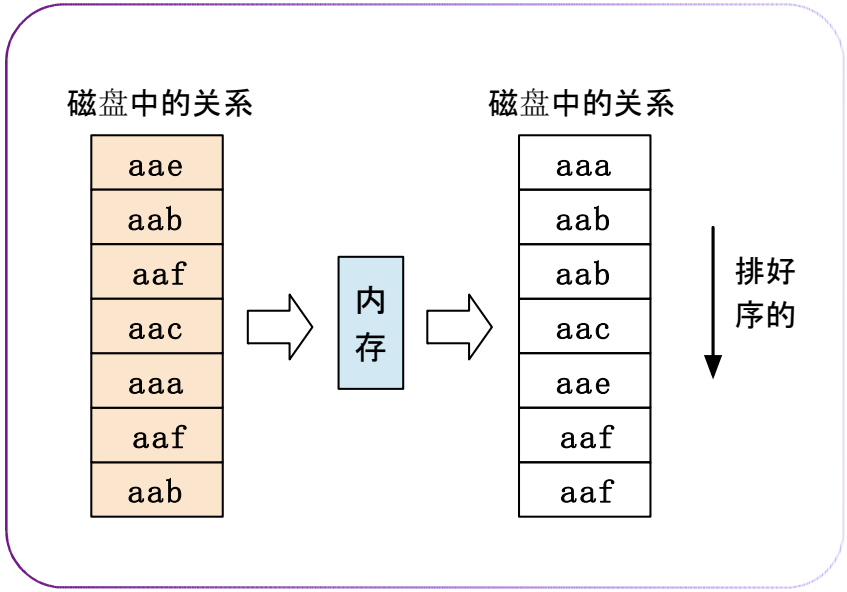
- **哈希的实现是指哈希连接中的哈希划分阶段**
  - 根据一个设定好的哈希函数将关系R 中的元组划分为子关系$R_0, R_1, \cdots R_K$
  - 对一个关系R进行哈希划分的代价为$2\mathrm{B}(R)$

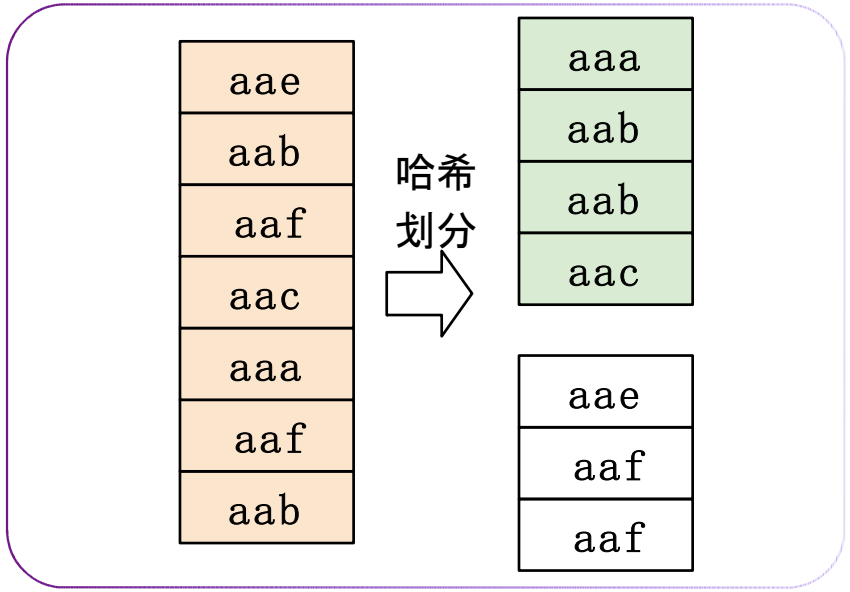




去重运算

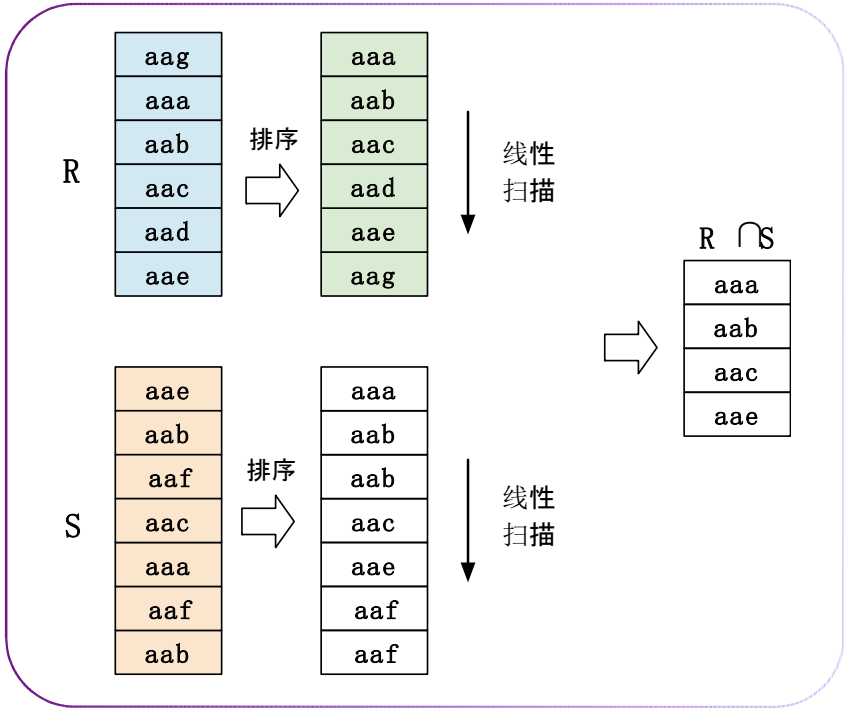
- 基于排序或哈希划分实现
- 在排序或哈希划分的过程中将重复的元组去除

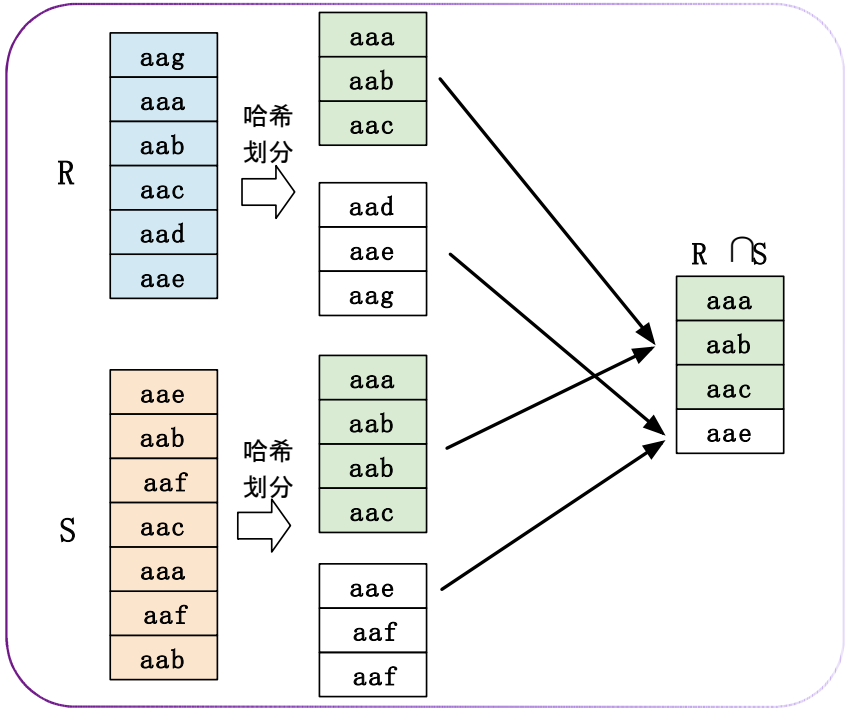




集合运算

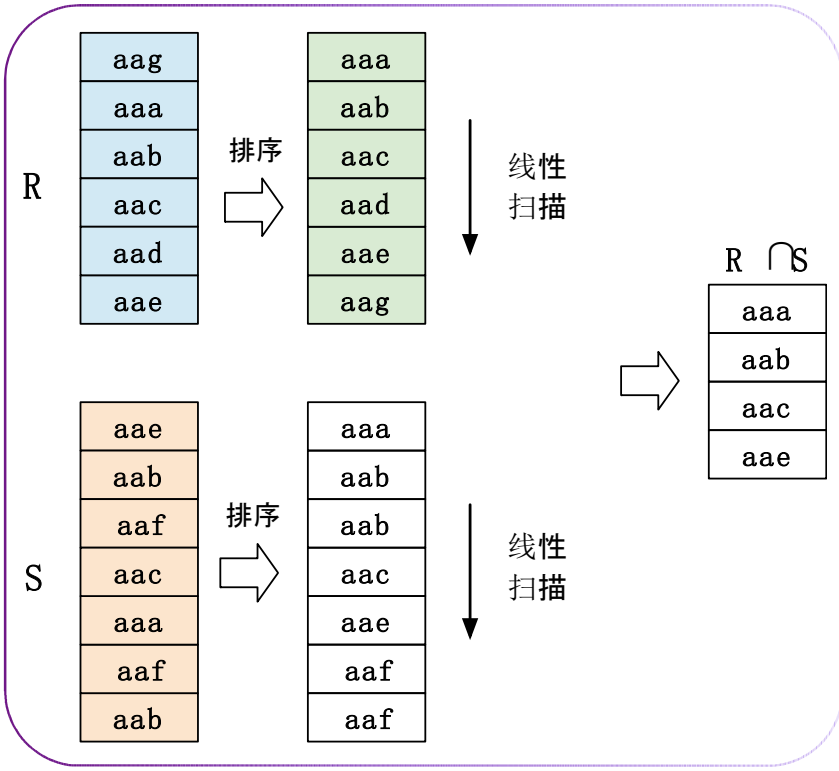
- 集合运算包含并（UNION）、交（INTERSECT）、差（DIFFERENCE）三种集合运算。
- 集合运算可以基于排序或哈希划分实现。





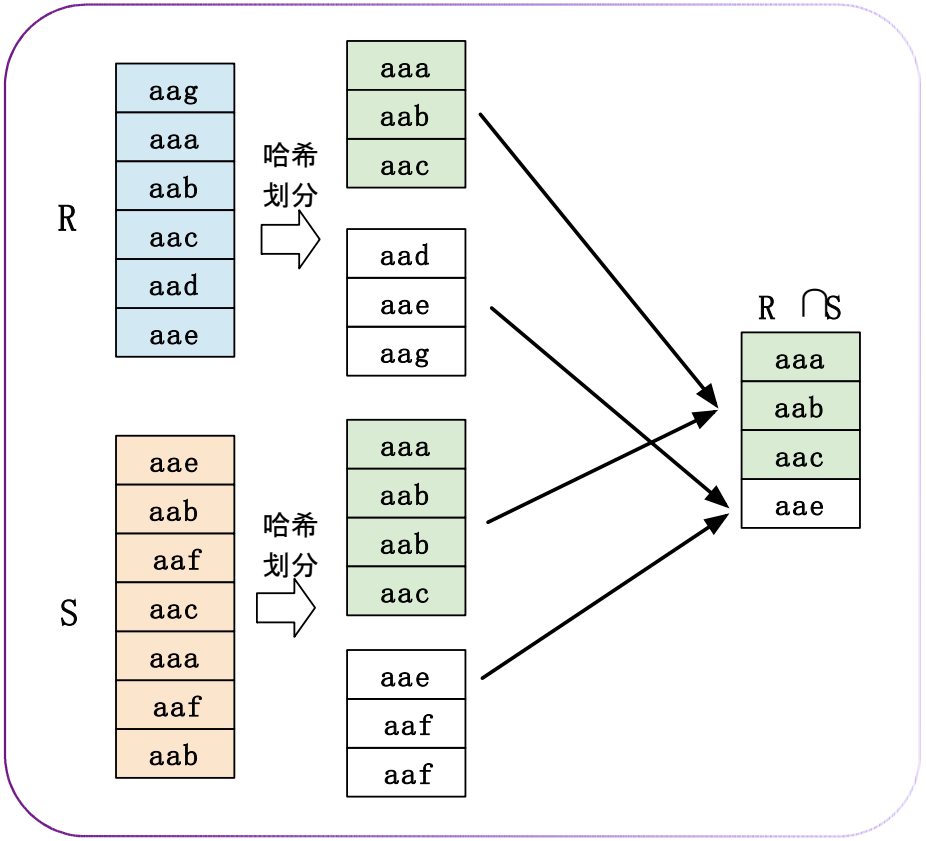
基于排序的集合运算

- **基于排序的实现：**
  - 将两个关系按相同的属性排序，之后对两个关系同时进行一次线性扫描，在线性扫描的过程中可以逐渐得到结果。
  - 对于并操作R ∪ S，可以通过同时扫描两个已排序的关系R和关系S，当两个关系中存在相同的元组时，合并结果中只保留其中一条元组。
  - 对于交操作R ∩ S，可以在合并结果中只保留那些出现在两个排序关系中的元组。
  - 对于差操作R − S，可以在合并结果中只保留 R 中但不存在于 S 中的元组。



基于哈希划分的集合运算

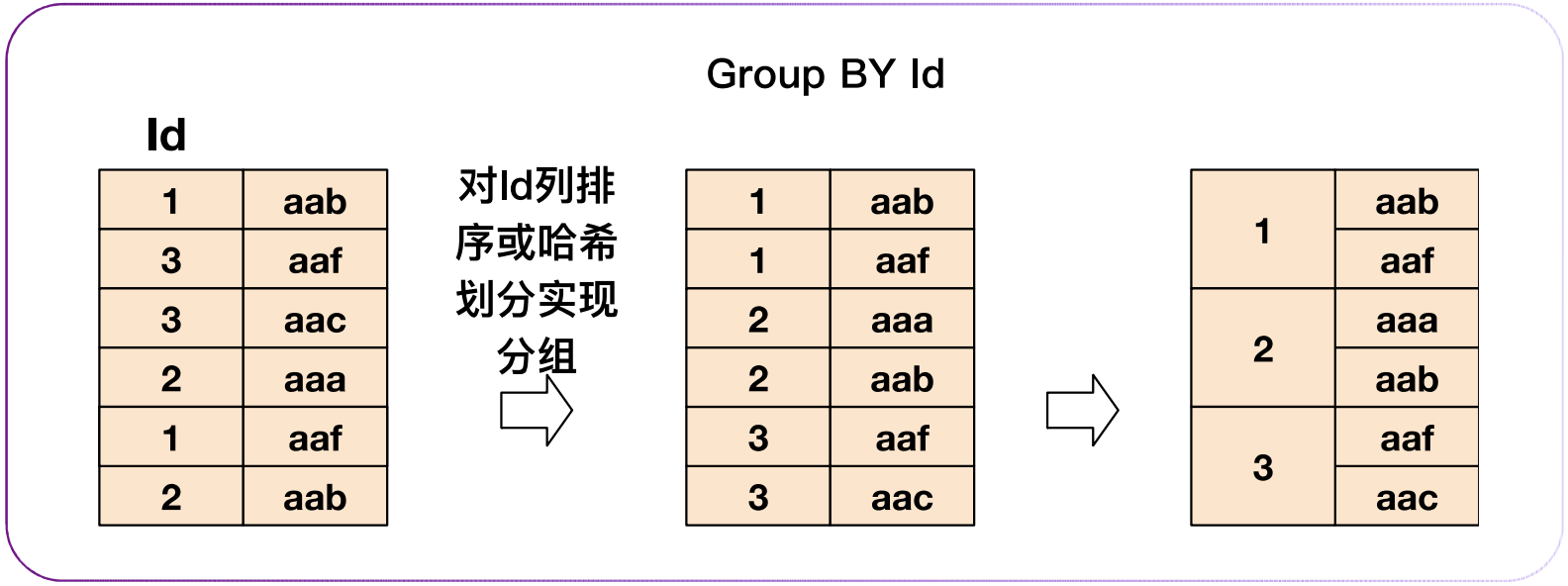
- **首先对R的元组进行哈希划分，之后对关系S中的元组逐个计算其哈希值，在计算哈希值的过程中实现集合运算。**
  - 实现R ∪ S：将与该哈希值对应的关系R的分区中的元组放入合并结果中，但不要放入重复的元组（放入结果前检查是否已放入该元组）；
  - 实现R ∩ S：将与该哈希值对应的关系R的分区中与该元组相同的一条元组放入交运算的结果中；
  - 实现R – S：判断对应分区中是否存在相同元组，若存在则在该分区中删除该元组，最终将所有分区中元组放入差运算的结果中即可。



聚集运算

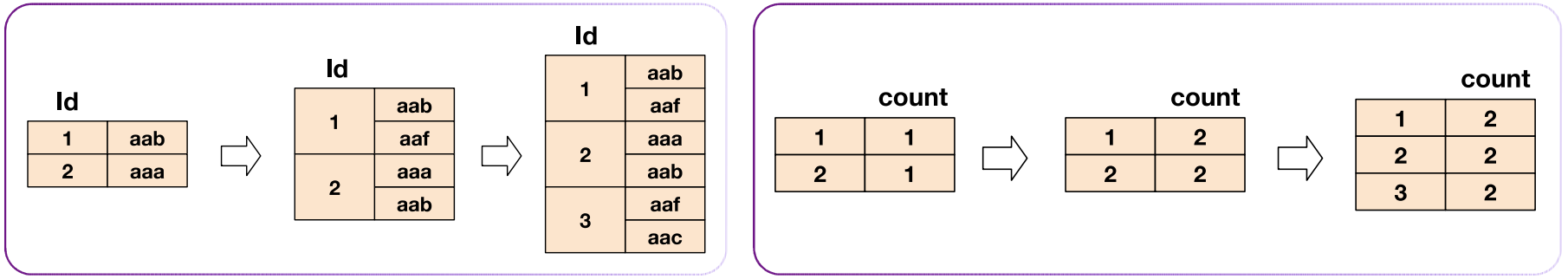
➢ 关系分组：

- 与去重运算的思路相同，可以使用排序或哈希划分实现关系的分组。
- 与去重不同的是：去重是为了去除关系中重复的元组，而关系分组是将关系中相同的元组组合为一个组。



聚集运算

- **聚集函数的实现方式如下：**
  - 不需要等到关系R中所有元组分组完成后进行聚集函数运算，可以在关系分组的过程中逐步实现 min、max、sum、count、avg 等聚集函数。
  - 比如对于 sum、min、max 运算，当分组过程中一组内出现两个元组时，可以提前计算目标元组并使用该元组替换这两个元组。
  - 对于count 运算，每组动态维护计数值即可。
  - 对于avg 运算，可以同时动态进行sum和count 两个聚集函数运算，在扫描完所有元组时将 sum 聚集函数的运算结果值除以 count 的结果值即可。



小结

- **本章概述了查询解析、查询优化与查询执行的概念。**
- **重点介绍了查询处理中查询解析器的工作原理、各查询算子的实现方式并分析了其磁盘I/O代价。**
  - 选择
  - 排序
  - 连接
  - 集合
  - 去重
  - 分组
- **查询优化与查询执行的具体细节将在第11、12章分别介绍**