9

# 第十章
# 关系查询处理及查询优化

李瑞轩
华中科技大学计算机学院

# 第十章　关系查询处理及查询优化

10.1 关系数据库系统的查询处理

10.2 关系数据库系统的查询优化

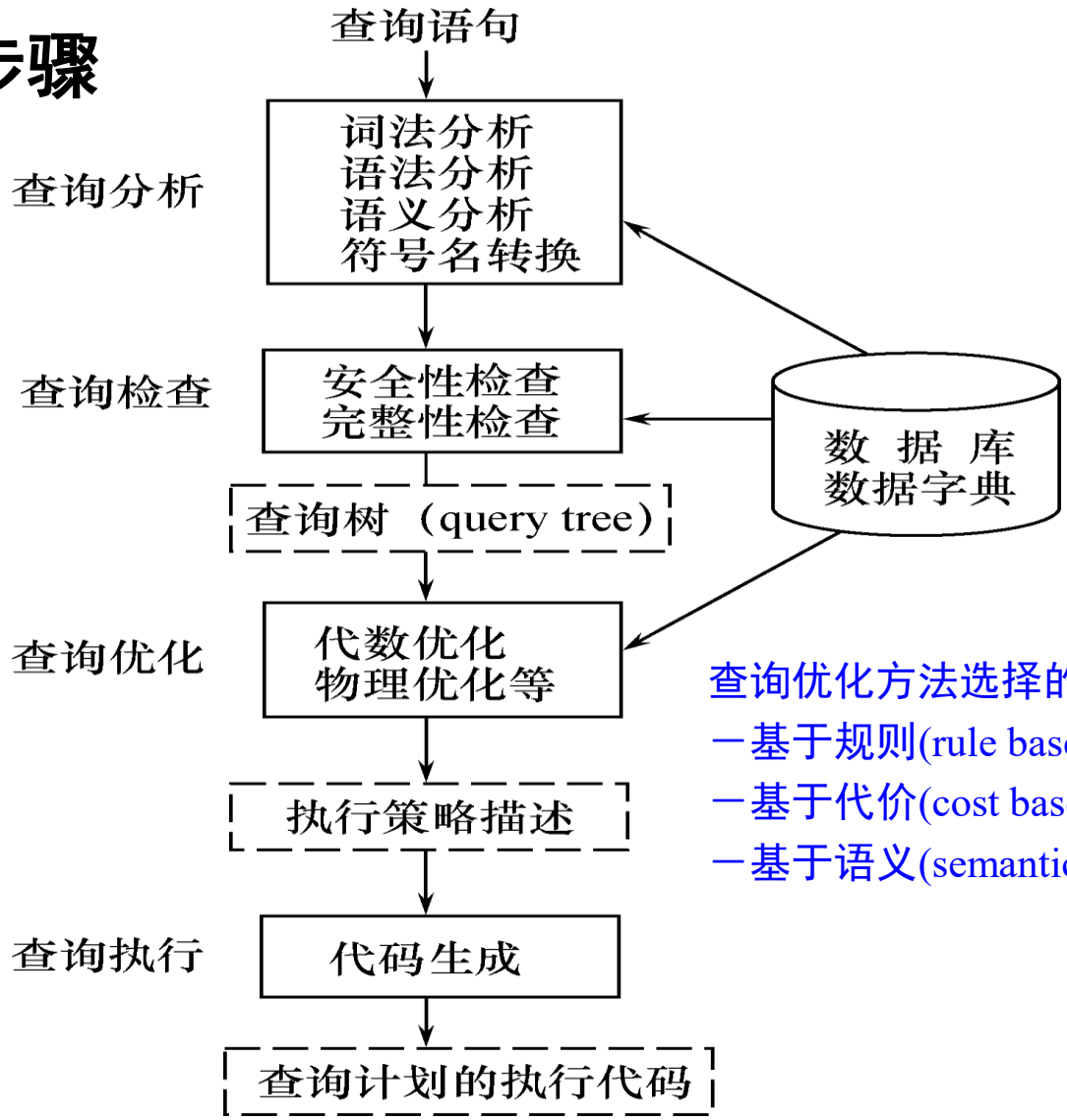
10.3 代数优化

10.4 物理优化

# 10.1 关系数据库系统的查询处理

## 10.1.1 查询处理步骤

1. 查询分析
2. 查询检查
3. 查询优化
4. 查询执行




查询优化方法选择的依据：

－基于规则(rule based)

－基于代价(cost based)

－基于语义(semantic based)

# 10.1.2 实现查询操作的算法示例

## 1. 选择操作的实现

［例1］ Select * from student where <条件表达式> ；

考虑<条件表达式>的几种情况：

C1：无条件；

C2：Sno＝'200215121'；

C3：Sage>20；

C4：Sdept＝'CS' AND Sage>20；

选择操作典型实现方法：

- ❖全表扫描（适合小表，不适合大表）；
- ❖索引（或散列）扫描（适合选择条件中的属性上有索引）

# 10.1.2 实现查询操作的算法示例

## 2. 连接操作的实现

- ［例2］ Select * from student， SC
  WHERE Student.Sno=SC.Sno；

连接操作典型实现方法：

1. 嵌套循环方法(nested loop)
2. 排序-合并方法(sort-merge join 或merge join)
3. 索引连接(index join)方法
4. Hash Join方法

# 10.1.2 实现查询操作的算法示例

- **2. 连接操作的实现**

  **嵌套循环方法**（nested loop）

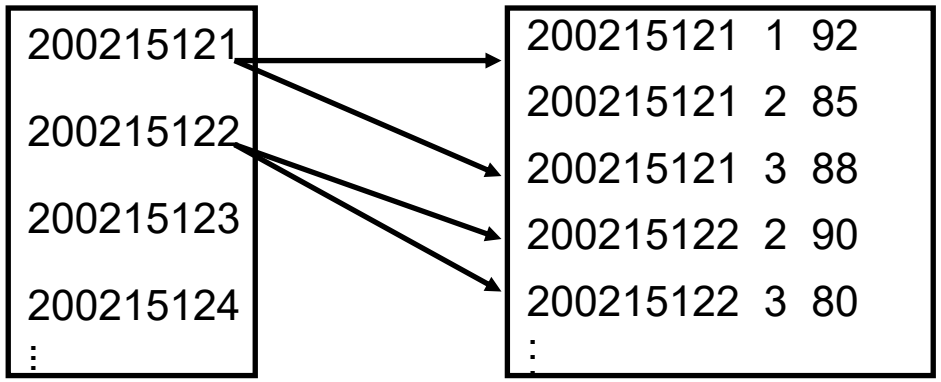
  - 对外层循环(Student)的每一个元组(s)，检索内层循环(SC)中的每一个元组(sc)
  - 检查这两个元组在连接属性(sno)上是否相等
  - 如果满足连接条件，则串接后作为结果输出，直到外层循环表中的元组处理完为止

# 10.1.2 实现查询操作的算法示例

- 2. 连接操作的实现

**排序-合并方法**（sort-merge join 或merge join）

- 适合连接的诸表已经排好序的情况
- 步骤：
  - 如果连接的表没有排好序，先对Student表和SC表按连接属性Sno排序
  - 取Student表中第一个Sno，依次扫描SC表中具有相同Sno的元组
  - ……




Student表和SC表都只要扫描一遍

  - 重复上述步骤直到Student 表扫描完

# 10.1.2 实现查询操作的算法示例

- **2. 连接操作的实现**

  **索引连接方法**（index join）

- 步骤：

  ① 在SC表上建立属性Sno的索引(如果原来没有)

  ② 对Student中每一个元组，由Sno值通过SC的索引查找相应的SC元组

  ③ 把这些SC元组和Student元组连接起来

  循环执行②③，直到Student表中的元组处理完为止

# 10.1.2 实现查询操作的算法示例

- 2.连接操作的实现

**Hash Join方法**

- 把连接属性作为hash码，用同一个hash函数把R和S中的元组散列到同一个hash文件中
- **步骤：**
  - 划分阶段(partitioning phase)：
    - 对包含较少元组的表(比如R)进行一遍处理
    - 把它的元组按hash函数分散到hash表的桶中
  - 试探阶段(probing phase)：也称为连接阶段(join phase)
    - 对另一个表(S)进行一遍处理
    - 把S的元组散列到适当的hash桶中
    - 把元组与桶中所有来自R并与之相匹配的元组连接起来

# 10.2 查询优化

## 10.2.1 查询优化概述

- 定义：为关系代数表达式的计算选择**最有效**的查询计划的过程。
- 目标：尽量提高关系数据库的存取效率
- 为什么要进行查询优化？
  - SQL语言的“非过程化”特性：不需要用户指明存取路径（Only What, Don't Why/How）；（查询优化的必要性）
  - 关系数据库的数据独立性：用户无法干预系统实现（查询优化的必要性）
  - 查询操作的多解性：同一个查询可能存在**多种效率不同**的实现方案。（查询优化的可能性）

# 10.2.1 查询优化概述

- 查询优化的优点：不仅在于用户不必考虑如何最好地表达查询以获得较好的效率，而且在于系统可以比用户程序的“优化”做得更好
  - (1) 优化器可以从数据字典中获取许多统计信息，而用户程序则难以获得这些信息
  - (2) 如果数据库的物理统计信息改变了，系统可以自动对查询重新优化以选择相适应的执行计划。在非关系系统中必须重写程序，而重写程序在实际应用中往往是不太可能的。
  - (3) 优化器可以考虑数百种不同的执行计划，程序员一般只能考虑有限的几种可能性。
  - (4) 优化器中包括了很多复杂的优化技术，这些优化技术往往只有最好的程序员才能掌握。系统的自动优化相当于使得所有人都拥有这些优化技术。

# 10.2.1 查询优化概述

- RDBMS通过某种代价模型计算出各种查询执行策略的执行代价，然后选取代价最小的执行方案
  - 集中式数据库
    - 执行开销主要包括：
      - 磁盘存取块数(I/O代价)
      - 处理机时间(CPU代价)
      - 查询的内存开销
    - **I/O代价是最主要的**
  - 分布式数据库
    - 总代价=I/O代价+CPU代价+内存代价＋通信代价

# 10.2.2 一个实例

［例］设有如下关系：

- 学生（学号，姓名，性别，出生日期，所在系）
- 课程（课号，名称，学分）
- 选课（学号，课号，成绩）

查询选修了2号课程的学生姓名。

```
SELECT  Student.Sname
    FROM  Student，SC
    WHERE  Student.Sno=SC.Sno AND
     SC.Cno='2';
```

可用如下等价的代数表达式来完成这一查询：

$Q_1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge Cno='2'}(Student \times SC))$

$Q_2 = \pi_{Sname}(\sigma_{Cno='2'}(Student \bowtie SC))$

$Q_3 = \pi_{Sname}(Student \bowtie \sigma_{Cno='2'}(SC))$

# 10.2.2 一个实例

- 假定有1000条学生记录，10000条选课记录，其中选修了2号课程的记录为50条。
- 设一个物理块能装10条学生记录或100条选课记录，内存提供了7块存储空间，其中5块存放学生记录，1块存放选课记录，1块用来存放中间结果，磁盘每秒钟读/写20块数据记录。

由于查询策略不同，这三种方案的查询时间相差很大。

❖**第一种方案：** $Q_1=\pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge Sc.Cno='2'}(Student \times SC))$

1) 计算笛卡尔积：学生×选课

读取总块数：1000/10 + 1000/(10×5)×10000/100 = 2100(块)

读取所需时间：T1 = 2100/20 = 105(秒)

学生记录块数

读选课记录遍数

选课记录块数

# 10.2.2 一个实例(续)

笛卡尔积运算结果的元组个数为：$10^3 \times 10^4 = 10^7$

设每块能装10个元组，则该结果集共需$10^6$块

写出所需时间：$T2 = 10^6/20 = 5 \times 10^4$(秒)

2) 作选择操作

依次读入笛卡尔积运算结果，选择满足条件的记录，假定内存处理时间忽略不计，则读取中间结果的时间T3与T2相等。即：$T3=5 \times 10^4$ (秒)

因为满足条件的记录仅有50条，可留在内存。

3) 作投影操作

将内存中的结果在“Sname”上作投影，得最终结果。内存处理时间忽略不计。

因此，方案一执行查询的总时间为：

$$T = T1+T2+T3 = 105 + 5 \times 10^4 + 5 \times 10^4 \approx 10^5 \text{(秒)}$$

27.8小时

## ❖第二种方案：$\pi_{Aname}(\sigma_{SC.cno='2'}(Student \bowtie SC))$

1) 计算自然连接：学生⋈选课

读取总块数与方案一相同，故所需时间也与方案一相同：

T1=105(秒)

但自然连接的运算结果只有$10^4$ 个元组（why？），设每块能装10个元组，则该结果集共需$10^3$ 块，

写出所需时间：$T2 = 10^3/20 = 50$ (秒)

2) 作选择操作：依次读入自然连接运算结果，选择满足条件的记录，假定内存处理时间忽略不计，则读取中间结果的时间T3与T2相等。即：T3=50 (秒)

因为满足条件的记录仅有50条，可留在内存。

3) 作投影操作：将内存中的结果在“Sname”上作投影，得最终结果。处理时间忽略不计。

因此，方案二执行查询的总时间为：

$T = T1+T2+T3 = 105+50+50 \approx 205$ (秒)

## ❖第三种方案：$\pi_{Sname}(Student \bowtie \sigma_{Cno='2'}(SC))$

1) 对选课表作选择操作

扫描一遍选课表，需要读取的块数为：10000/100＝100(块)

花费时间为：T1＝100/20 = 5 (秒)

因满足条件的记录为50条，不必写磁盘。

2) 作自然连接

读取学生表，与内存中的选课记录作连接，需要读取的块数为：1000/10＝100(块)，连接结果不必写磁盘。

花费时间为：T2＝100/20 = 5 (秒)

3) 输出投影结果

在内存中操作，处理时间忽略不计。

因此，方案三执行查询的总时间为：

$$T = T1+T2＝5＋5 \approx 10 \text{(秒)}$$

由$10^5 \rightarrow 205 \rightarrow 10$，本例充分说明了查询优化的必要性，同时也给出了一些查询优化方法的初步概念。

# 10.2.2 一个实例(续)

- 假如SC表的Cno字段上有索引
  - 第一步就不必读取所有的SC元组而只需读取Cno='2'的那些元组(50个)
  - 存取的索引块和SC中满足条件的数据块大约总共3～4块
- 若Student表在Sno上也有索引
  - 第二步也不必读取所有的Student元组
  - 因为满足条件的SC记录仅50个，涉及最多50个Student记录
  - 读取Student表的块数也可大大减少
- 总的存取时间将进一步减少到数秒

# 10.2.2 一个实例(续)

$Q_1 = \pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge Cno='2'}(Student \times SC))$

$Q_2 = \pi_{Sname}(\sigma_{Cno='2'}(Student \bowtie SC))$

$Q_3 = \pi_{Sname}(Student \bowtie \sigma_{Cno='2'}(SC))$

- 把代数表达式$Q_1$变换为$Q_2$、 $Q_3$，
  - 即有选择和连接操作时，先做选择操作，这样参加连接的元组就可以大大减少，这是代数优化
- 在$Q_3$中，
  - SC表的选择操作算法有全表扫描和索引扫描2种方法，经过初步估算，索引扫描方法较优
  - 对于Student和SC表的连接，利用Student表上的索引，采用index join代价也较小，这就是物理优化

# 10.3 代数优化

关系代数表达式等价转换规则：

- 笛卡儿积、自然连接、并、交的交换律和结合律
  - R×S＝S×R；　(R×S)×T＝R×(S×T)
  - R⋈S＝S⋈R；　(R⋈S)⋈T＝R⋈(S⋈T)
  - R∪S＝S∪R；　(R∪S)∪T＝R∪(S∪T)
  - R∩S＝S∩R；　(R∩S)∩T＝R∩(S∩T)

R

| A | B |
|---|---|
| 10 | 20 |
| 20 | 30 |
| 30 | 40 |

S

| A | C |
|---|---|
| 10 | 20 |
| 20 | 30 |
| 30 | 40 |

T

| C | D |
|---|---|
| 20 | 20 |
| 10 | 30 |
| 10 | 40 |

(R⋈S)⋈T：中间结果R⋈S产生3条记录
R⋈(S⋈T)：中间结果S⋈T产生1条记录

查询代价不同

# 代数优化

- **选择上的转换规则**
  - $\sigma_{F1 \wedge F2}(R) = \sigma_{F1}(\sigma_{F2}(R))$
  - $\sigma_{F1 \vee F2}(R) = (\sigma_{F1}(R)) \cup (\sigma_{F2}(R))$
- **选择＋自然连接**

  设选择条件 p 只涉及到关系R的属性；条件 q 只涉及到关系S的属性；条件m既涉及关系R又涉及关系S，则下列等式成立：
  - $\sigma_p(R \bowtie S) = (\sigma_p(R)) \bowtie S$
  - $\sigma_q(R \bowtie S) = R \bowtie (\sigma_q(S))$
  - $\sigma_{p \wedge q}(R \bowtie S) = (\sigma_p(R)) \bowtie (\sigma_q(S))$
  - $\sigma_{p \wedge q \wedge m}(R \bowtie S) = \sigma_m((\sigma_p(R)) \bowtie (\sigma_q(S)))$
  - $\sigma_{p \vee q}(R \bowtie S) = ((\sigma_p(R)) \bowtie S) \cup (R \bowtie (\sigma_q(S)))$

# 代数优化

- **投影＋选择**

  设X为关系R的一个属性集；Z为条件F中所涉及到的关系R的属性集，则：

  - 当$Z \in X$时：

    $$\pi_X(\sigma_F(R)) = \sigma_F(\pi_X(R))$$

  - 当$Z \notin X$时：

    $$\pi_X(\sigma_F(R)) = \pi_X(\sigma_F(\pi_{XZ}(R)))$$

- **投影＋自然连接**

  设X为关系R的一个属性集；Y为关系S的一个属性集，Z为关系R和S的交集，则：

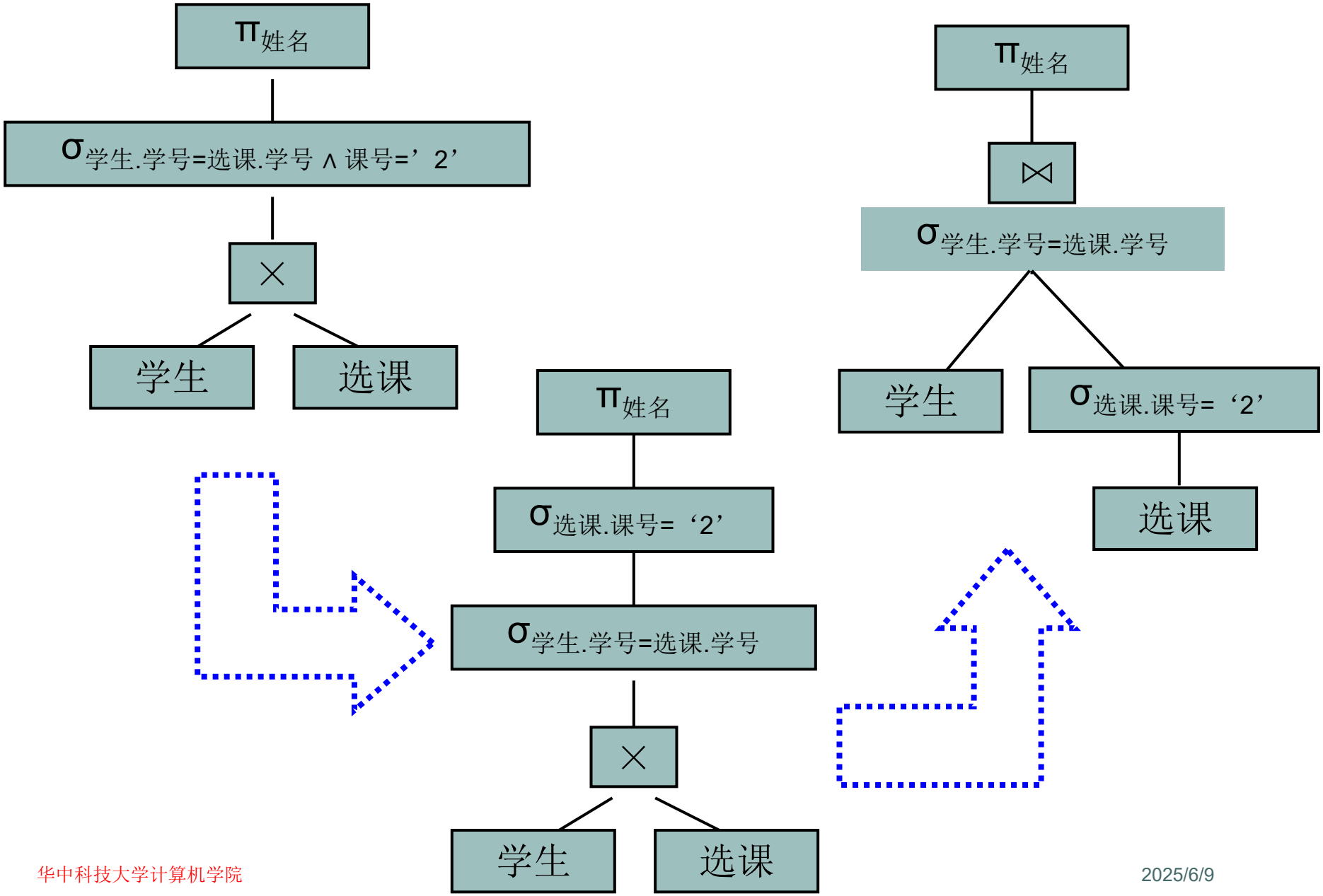
  $$\pi_{XY}(R \bowtie S) = \pi_{XY}((\pi_{XZ}(R)) \bowtie (\pi_{YZ}(S)))$$

# 10.3.2 查询树的启发式优化

- 转换的最终目的
  - 减少查询的开销（I/O次数）
- 转换的直接目的
  - 减少中间关系的元组数、元组大小

- **规则1**：选择运算尽早执行（减小中间结果集的规模）；
- **规则2**：将选择、投影运算同时进行（避免重复扫描）；
- **规则3**：把投影同其前或其后的双目运算结合（避免重复扫描）；
- **规则4**：把选择运算与其前面的笛卡尔积运算结合起来一起计算；（避免重复扫描并减小中间结果集的规模）
- **规则5**：把公共子表达式的运算结果存放在外存，作中间结果，使用时读入主存，如视图表达式（避免重复计算）

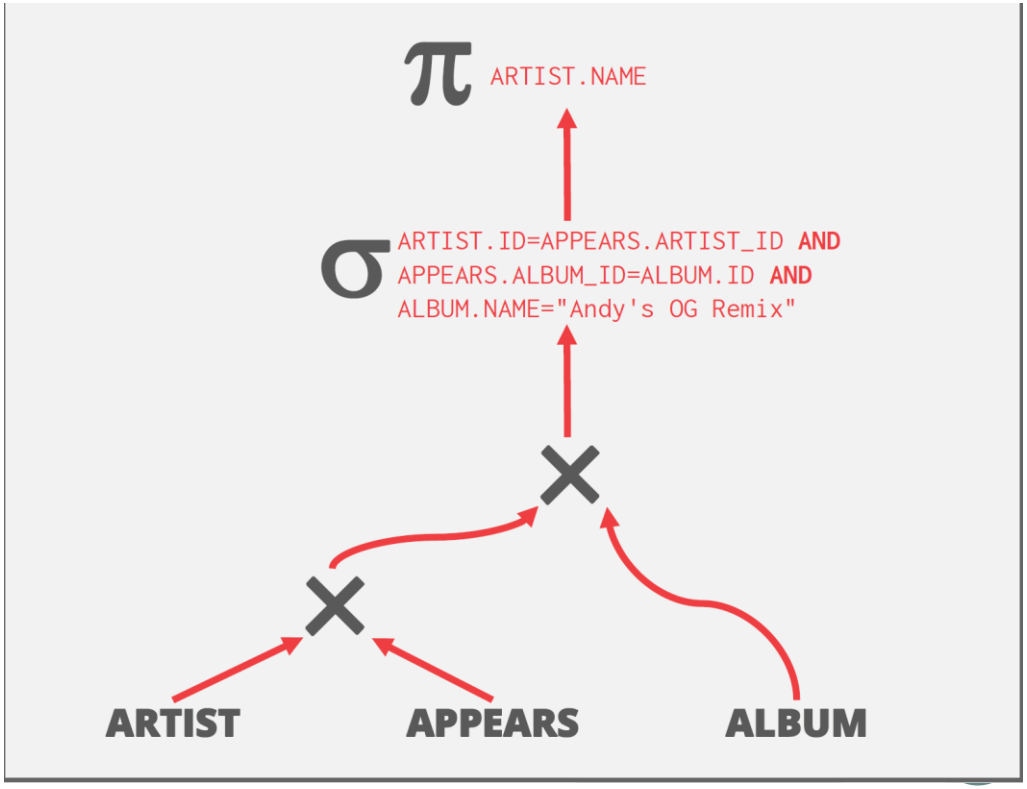
# 例1：查询树的优化


π姓名
σ学生.学号=选课.学号 ∧ 课号=’2’
×
学生
选课
π姓名
σ选课.课号=‘2’
σ学生.学号=选课.学号
×
学生
选课
π姓名
⋈
σ学生.学号=选课.学号
学生
σ选课.课号=‘2’
选课

# 例2：查询树的优化

```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Andy's OG Remix"
```

拆分选择条件

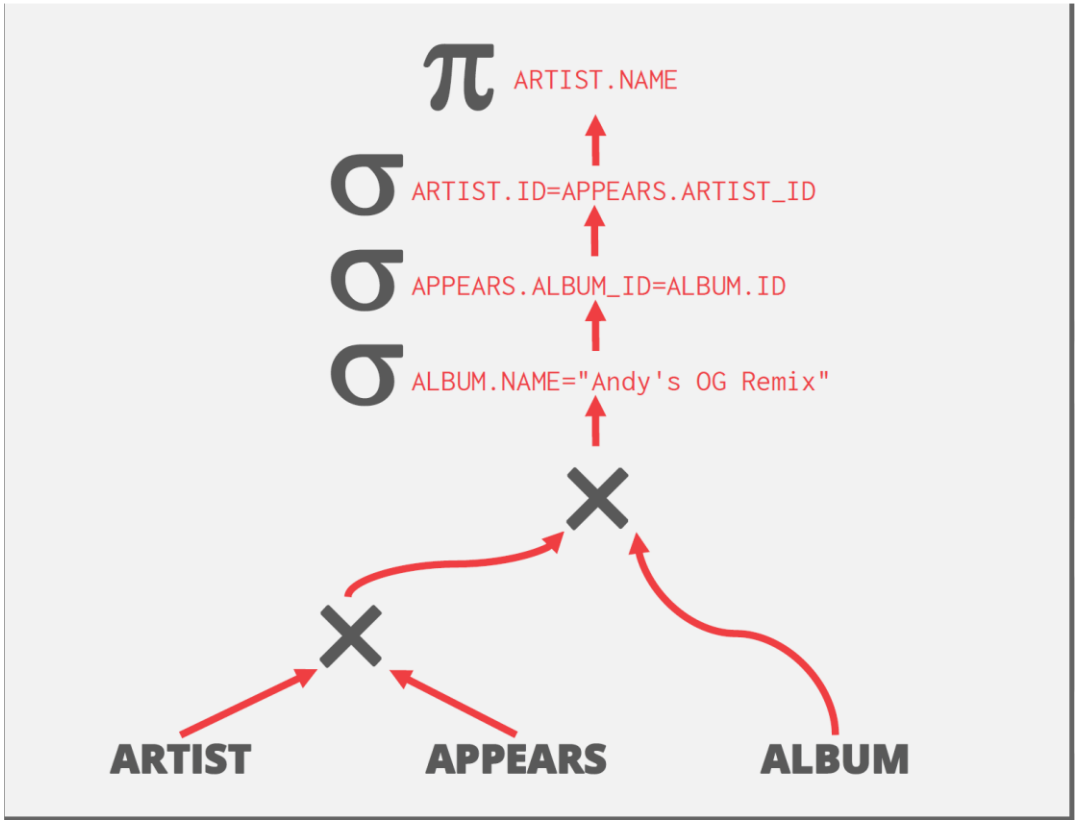


# 例2：查询树的优化

```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Andy's OG Remix"
```

将选择尽量下推到

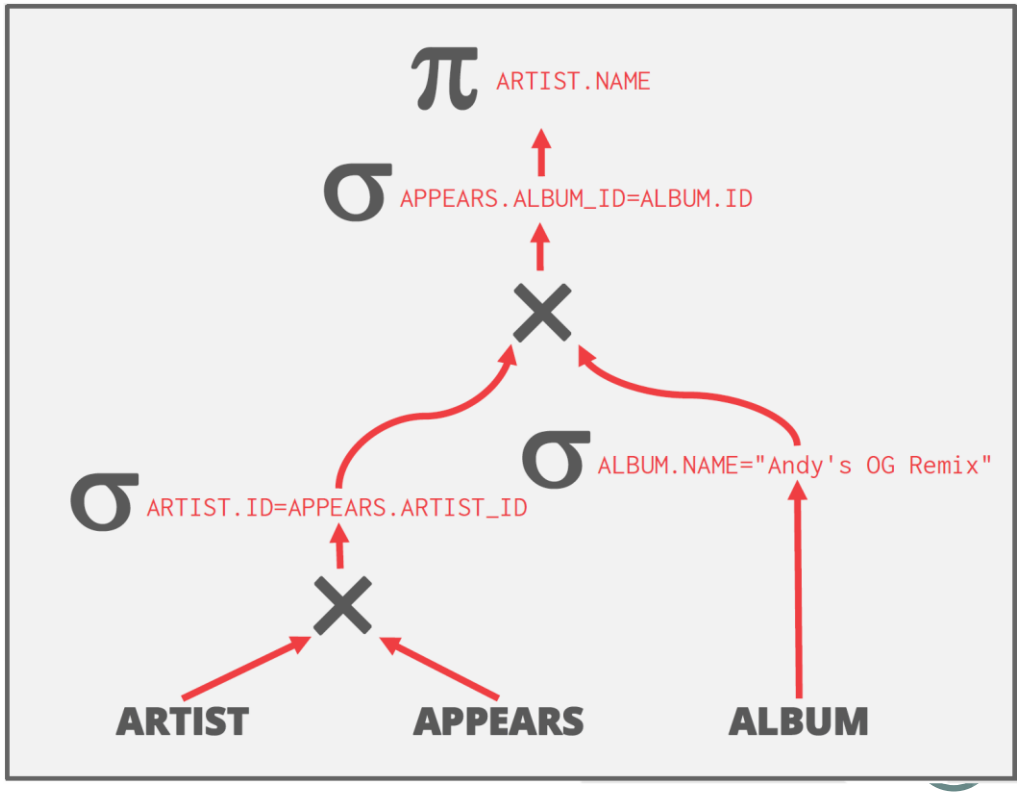
笛卡尔积之前进行



# 例2：查询树的优化

```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Andy's OG Remix"
```

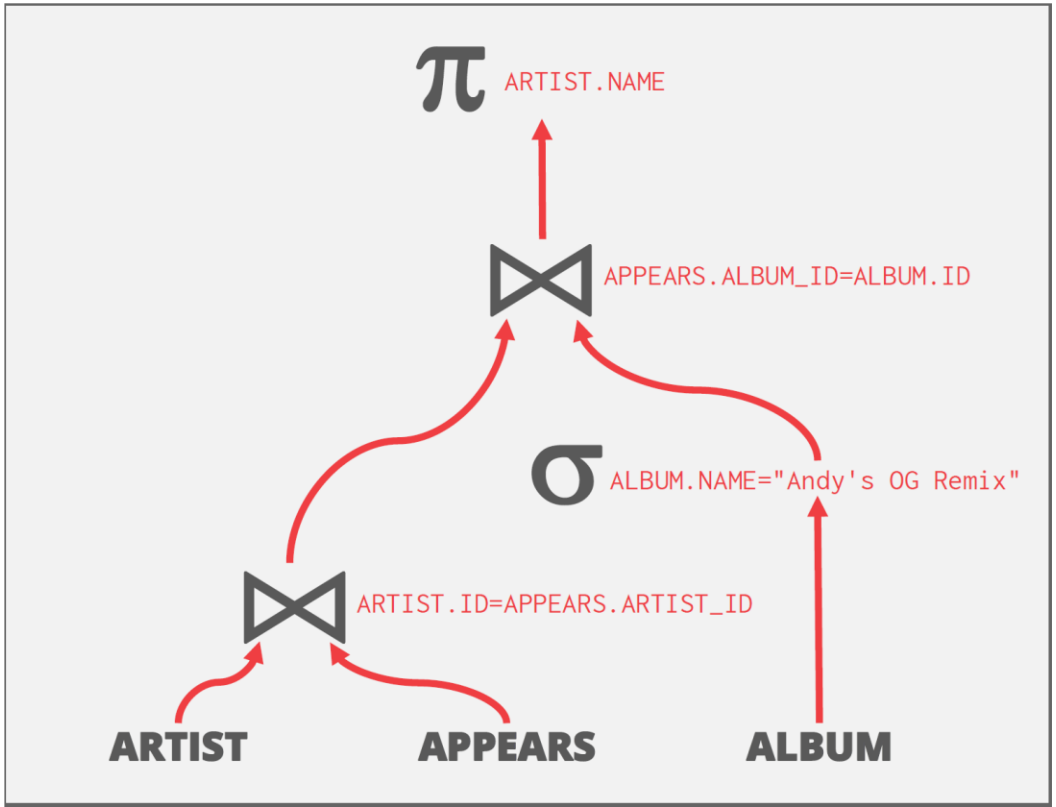
把选择运算与其前面的笛卡尔积运算结合起来，用内连接代替笛卡尔积。



# 例2：查询树的优化

```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Andy's OG Remix"
```

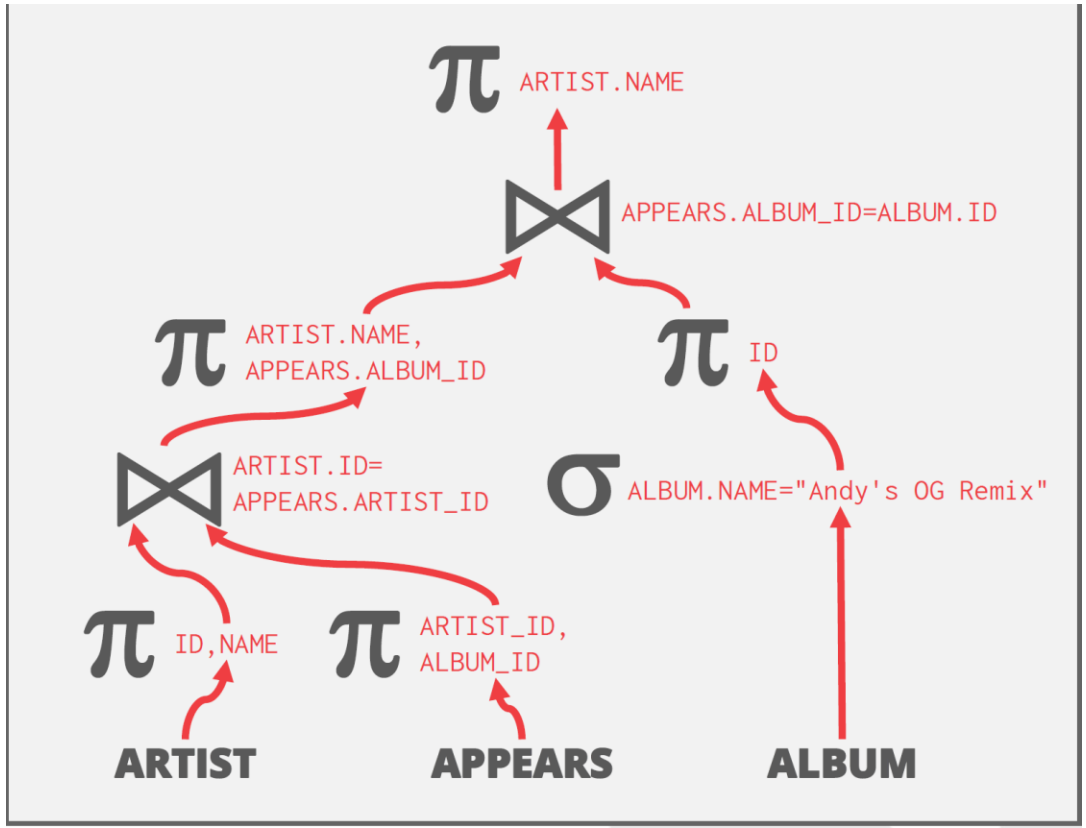
把选择运算与其前面的笛卡尔积运算结合起来，用内连接代替笛卡尔积。



# 例2：查询树的优化

```
SELECT ARTIST.NAME
  FROM ARTIST, APPEARS, ALBUM
 WHERE ARTIST.ID=APPEARS.ARTIST_ID
   AND APPEARS.ALBUM_ID=ALBUM.ID
   AND ALBUM.NAME="Andy's OG Remix"
```

消去不必要的属性，减小中间结果集的规模。



# 10.4 物理优化

- 代数优化改变查询语句中操作的次序和组合，不涉及底层的存取路径。
- 对于一个查询语句有许多存取方案，它们的执行效率不同， 仅仅进行代数优化是不够的。
- 物理优化就是要选择高效合理的操作算法或存取路径，求得优化的查询计划。

- **选择的方法：**
  - 基于规则的启发式优化
  - 基于代价估算的优化
  - 两者结合的优化方法

# 10.4.1 基于规则的优化

## 1. 选择操作的启发式规则:

1) 对于小关系，使用全表扫描；
2) 对于大关系：
   - 如果是主码值上的等值查询，使用主码索引；
   - 如果是非主码值上的等值查询，在选择比例<10%的情况下，使用索引（如果有）；否则使用全表扫描；
   - 如果非等值或范围查询，在选择比例<10%的情况下，使用索引（如果有）；否则使用全表扫描；
   - 对于用AND连接的合取选择条件，如果有涉及这些属性的组合索引，优先采用组合索引扫描方法；如果某些属性上有一般的索引，则可以用索引扫描方法；否则使用全表顺序扫描。
   - 对于用OR连接的析取选择条件，一般使用全表顺序扫描

# 10.4.1 基于规则的优化

**2. 连接操作的启发式规则：**

1）如果2个表都已经按照连接属性排序

- 选用排序-合并方法

2）如果一个表在连接属性上有索引

- 选用索引连接方法

3）如果上面2个规则都不适用，其中一个表较小

- 选用Hash join方法

4）可以选用嵌套循环方法，并选择其中较小的表，确切地讲是占用的块数(b)较少的表，作为外表(外循环的表)。

# 10.4.1 基于规则的优化

- 选择嵌套循环方法时，为什么要选择占用块数(b)较少的表作为外表？
  - 设连接表R与S分别占用的块数为Br与Bs
  - 连接操作使用的内存缓冲区块数为K
  - 分配K-1块给外表
  - 如果R为外表，则嵌套循环法存取的块数为
    Br+BrBs/(K-1)　//数据缓冲区大小为K块，且K<Br<Bs
  - 显然应该选块数小的表作为外表

# 10.4.2 基于代价的优化

- 启发式规则优化是定性的选择，适合解释执行的系统
  - 解释执行的系统，优化开销包含在查询总开销之中
- 编译执行的系统中查询优化和查询执行是分开的
  - 可以采用精细复杂一些的基于代价的优化方法

# 10.4.2 基于代价的优化

## 统计信息：

- 基于代价的优化方法要计算各种操作算法的执行代价，与数据库的状态密切相关
- 数据字典中存储的优化器需要的统计信息：

1. 对每个基本表
   - 该表的元组总数(N)
   - 元组长度(l)
   - 占用的块数(B)
   - 占用的溢出块数(BO)

2. 对基表的每个列
   - 该列不同值的个数(m)
   - 选择率(f)
     - 如果不同值的分布是均匀的，f＝1/m
     - 如果不同值的分布不均匀，则每个值的选择率＝具有该值的元组数/N
   - 该列最大值
   - 该列最小值
   - 该列上是否已经建立了索引
   - 索引类型(B+树索引、Hash索引、聚集索引)

3. 对索引(如B+树索引)
   - 索引的层数(L)
   - 不同索引值的个数
   - 索引的选择基数S(有S个元组具有某个索引值)
   - 索引的叶结点数(Y)

# 10.4.2 基于代价的优化

## 1. 中间结果的大小估计

- 需要使用一些统计数据(statistics)
  - T(R) : R的元组数
  - S(R) : R中每个元组的大小(bytes)
  - V(R, A) : R的属性A上的不同值数

R

| A | B | C | D |
|---|---|---|---|
| cat | 1 | 10 | a |
| cat | 1 | 20 | b |
| dog | 1 | 30 | a |
| dog | 1 | 40 | c |
| bat | 1 | 50 | d |

A: 20 byte；B: 4 byte；C: 8 byte；D: 5 byte

关于R的统计数据：

T(R) = 5　　S(R) = 37

V(R,A) = 3　　V(R,C) = 5

V(R,B) = 1　　V(R,D) = 4

关于中间结果W = R1×R2的大小估计如下：

T(W) = T(R1)×T(R2)

S(W) = S(R1) + S(R2)

# 基于代价的优化

$W = \sigma_{D=a}(R)$的大小估计：

S(W) = S(R)

T(W) = ?

R

| A | B | C | D |
|---|---|---|---|
| cat | 1 | 10 | a |
| cat | 1 | 20 | b |
| dog | 1 | 30 | a |
| dog | 1 | 40 | c |
| bat | 1 | 50 | d |

- 假设D上的值在V(R,D)个不同值上均匀分布：

V(R,A) = 3；V(R,C) = 5；V(R,B) = 1；V(R,D) = 4

T(W) = T(R) / V(R,D)

# 基于代价的优化

选择大小的估计 $W = \sigma_p(R)$

$$T(W) = s * T(R)$$

其中 s 是选择率。

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

# 选择率的估计

- 选择率（selectivity）
  - 谓词P的选择率指的是复合谓词P的元组占有率。
- 选择率依赖于谓词P的类型
  - 相等
  - 范围
  - 非
  - 与
  - 或

# 选择率的估计

假设V(age,people)具有5个不同的值（18-22）且$N_R$=5。

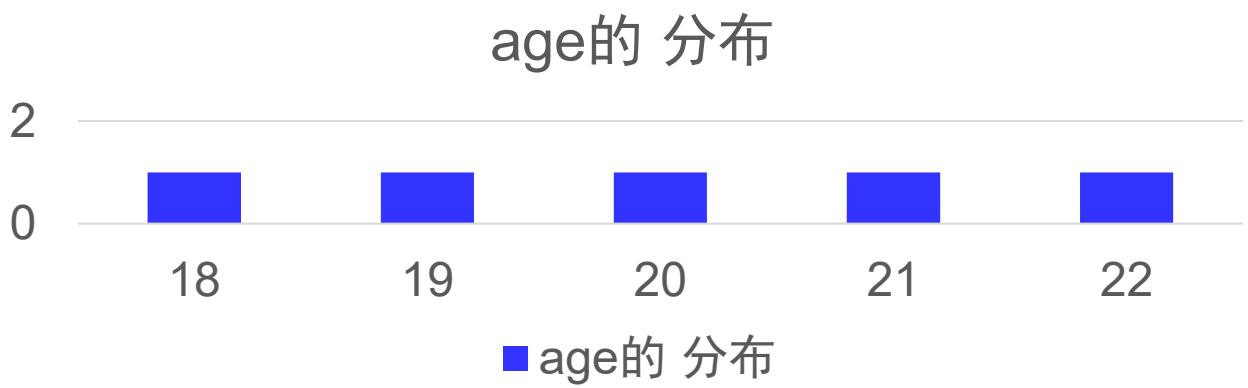
- **相等谓词**

  Select * from People where age=20

  相等谓词：A=constant

  相等谓词的选择率：Sel(A=constant)=1/ V(R,A)

  Sel(age=20)=1/5



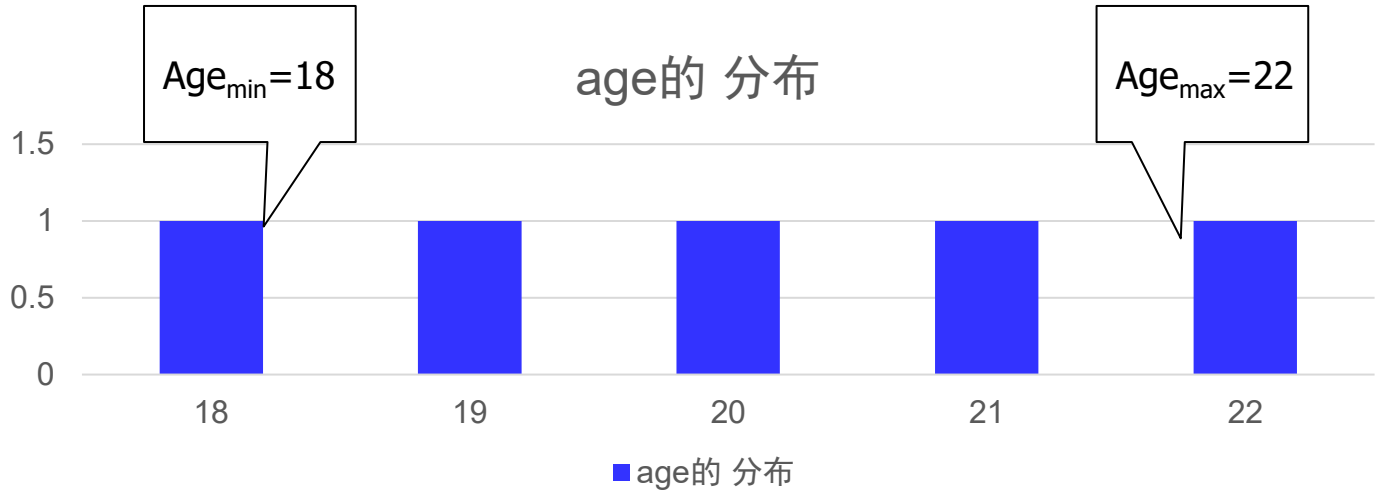
# 选择率的估计

- 范围谓词

$Sel(A>=a)=(A_{max}-a+1)/((A_{max}-A_{min}+1))$

**Select * from People where age>=20**

Sel(age>=20)=(22-20+1)/(22-18+1)=3/5



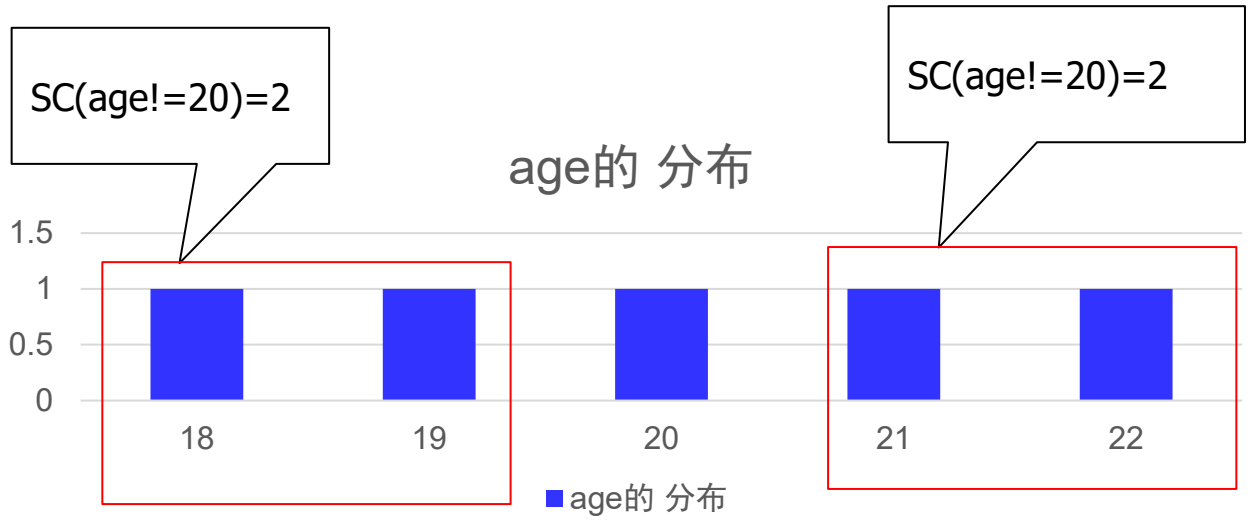
# 选择率的估计

- 非谓词

Sel(!P)=1-sel(P)

**Select * from People where age!=20**

Sel(age!=20)=1-1/5=4/5



# 选择率的估计

- 与谓词

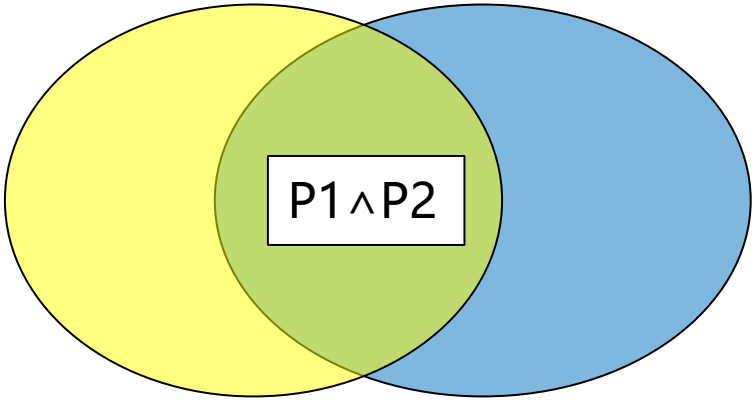
  当谓词之间的选择独立时

  Sel(P1∧P2)=sel(P1)×sel(P2)

  查询：

  **Select * from People**

  **where age=20 and name like ’王%’**



# 选择率的估计

- **或谓词**

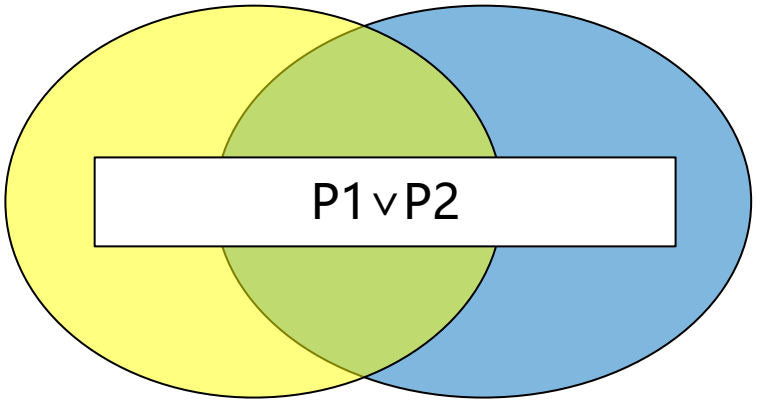
当谓词之间的选择独立时

$$Sel(P1 \vee P2) = sel(P1) + sel(P2) - Sel(P1 \wedge P2)$$

$$= sel(P1) + sel(P2) - Sel(P1) \times Sel(P2)$$

**查询：Select * from People**

**where age=20 or name like ’王%’**



# 选择率的估计

- 采样估计
  - 从大表中进行采样，通过采样表估计选择率。

```
Select AVG(age)
From people
Where age>50
```

Sel(age>=50)

=1/3

采样表

| id | name | age | staus |
|---|---|---|---|
| 1001 | Zhao | 59 | 退休 |
| 1003 | Sun | 25 | 旷工 |
| 1005 | Zhou | 39 | 休假 |

| id | name | age | staus |
|---|---|---|---|
| 1001 | Zhao | 59 | 退休 |
| 1002 | Qian | 41 | 正常 |
| 1003 | Sun | 25 | 旷工 |
| 1004 | Li | 26 | 离职 |
| 1005 | Zhou | 39 | 休假 |
| 1006 | Wu | 57 | 正常 |

...10亿条元组

# 基于代价的优化

## 2. I/O代价估计

- 估计执行查询计划所必须读（写）的磁盘块数目，需要一些参数：
  - B(R)：R所需的块数
  - f(R)：每块可容纳的R的最大元组数
  - M：可用的内存块数
  - HT(i)：索引i的层数
  - LB(i)：索引i的叶结点所需的块数
- 影响查询计划I/O代价的因素：
  - 实现查询计划的逻辑操作符
  - 操作的顺序：例如，多关系的连接顺序
  - 中间结果的大小
  - 实现逻辑操作符的物理操作符：例如，连接操作是用索引连接还是散列连接？

# I/O代价的估算

- **全表扫描**
  - 访问所有数据：cost=B(R)　//R占用的磁盘块数(B)
  - 主码列等值查找：　cost=B(R)/2
- **B+树索引扫描**
  - 主码列等值查找：cost=H(i)+1　//B+树索引的高度(H)
  - 非主码列等值查找（最坏情况）：cost=H(i)+S　//S为满足条件的元组数
- **嵌套循环连接 W=R⋈S**
  - 读取总块数：B(R)+B(R)B(S)/(K-1)　//数据缓冲区大小为K块，且K<B(R)<B(S)
  - 写出总块数：T(W) / M(W)　//元组总数(T)，块因子(M)

# 小结

- 查询处理是RDBMS的核心，查询优化技术是查询处理的关键技术
- 本章讲解的优化方法
  - 启发式代数优化
  - 基于规则的存取路径优化
  - 基于代价的优化
- 比较复杂的查询，尤其是涉及连接和嵌套的查询
  - 不要把优化的任务全部放在RDBMS上
  - 应该找出RDBMS的优化规律，以写出适合RDBMS自动优化的SQL语句

# 本章作业：P290：2，3