2024年春季学期

数据库系统概论

An Introduction to Database Systems

第九章 关系查询处理和查询优化

中国科学技术大学 大数据学院

黄振亚，huangzhy@ustc.edu.cn

第三篇 系统篇

- 讨论数据库管理系统中查询处理和事务管理的基本概念与基础知识
  - 第9章 关系查询处理和查询优化
  - 第10章 数据库恢复技术
  - 第11章 并发控制

第九章 关系系统及其查询优化

关系系统及其查询优化（续）

- **本章目的：**
  - RDBMS的查询处理步骤
  - 查询优化的概念
  - 基本方法和技术

- 查询优化分类：
  - 代数优化
  - 物理优化

9.1 关系数据库系统的查询处理

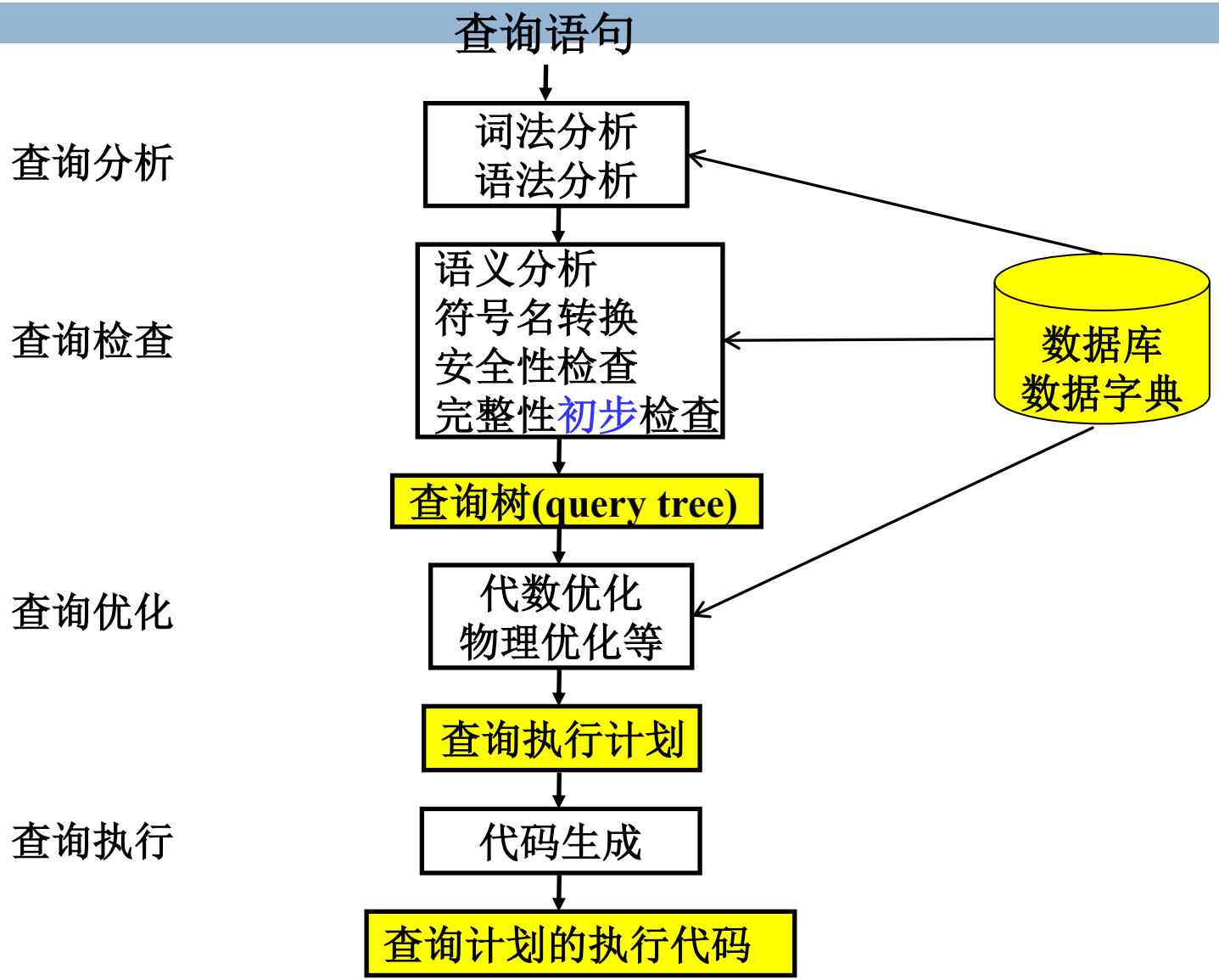
- **9.1.1 查询处理步骤**
- **9.1.2 实现查询操作的算法示例**

9.1.1 查询处理步骤

- **RDBMS查询处理阶段：**

1. 查询分析
2. 查询检查
3. 查询优化
4. 查询执行

查询处理步骤


查询语句
查询分析
词法分析
语法分析
查询检查
语义分析
符号名转换
安全性检查
完整性初步检查
数据库
数据字典
查询树(query tree)
查询优化
代数优化
物理优化等
查询执行计划
查询执行
代码生成
查询计划的执行代码

SQL语句

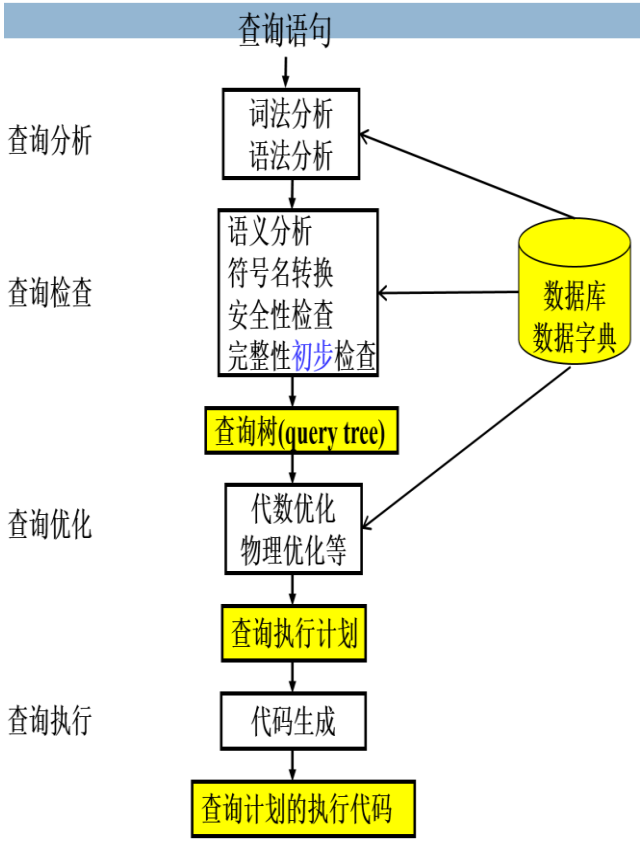
- Select sno, sname, cno, grade
- From Student, SC
- Where Student.sno = SC.sno and grade < 60

1. 查询分析

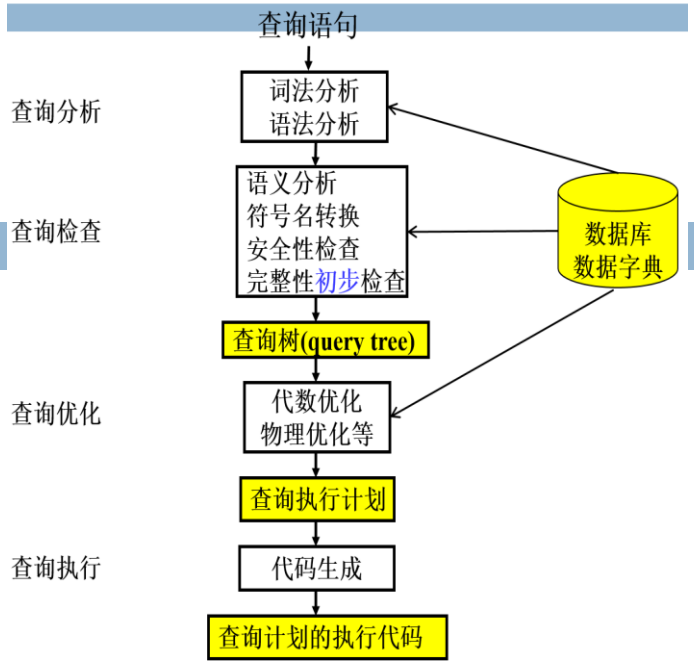
- 对查询语句进行扫描、词法分析和语法分析

- 词法分析：从查询语句中识别出语言符号
  - SQL关键字、属性名、关系名等

- 语法分析：进行语法检查和语法分析
  - 是否符合SQL语法规则



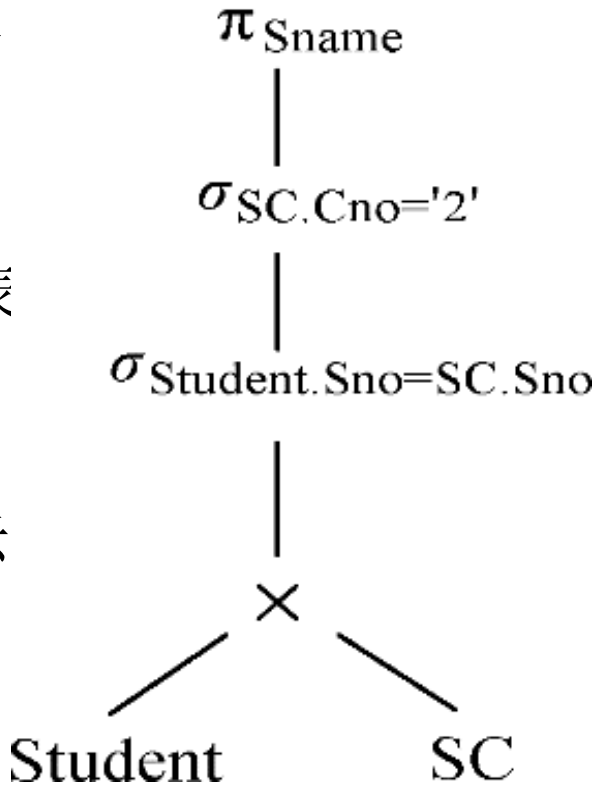
2. 查询检查



- 查询检查的任务
  - 合法权检查
  - 视图转换
  - 安全性检查：用户权限
  - 完整性初步检查（静态）
- 目标：根据数据字典对合法的查询语句进行语义检查
  - 数据库对象，如关系名、属性名、视图名是否存在且有效
  - 若对视图的操作，则进行视图消解，或实体化视图
- 根据数据字典中的用户权限和完整性约束定义对用户的存取权限进行检查
  - 是否需要拒绝执行？
    - 安全性：权限
    - 完整性：如，grade是int型

2. 查询检查

- 检查通过后，把SQL查询语句转换成等价的关系代数表达式
- RDBMS一般都用查询树(语法分析树)来表示扩展的关系代数表达式
- 把数据库对象的外部名称转换为内部表示

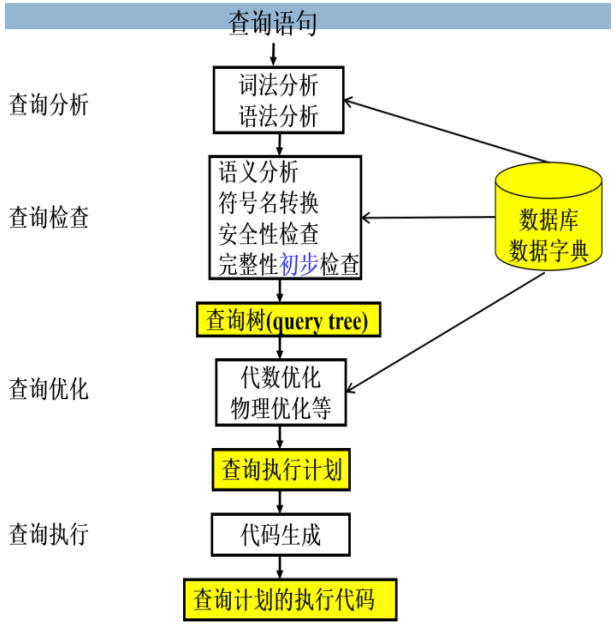


关系代数语法树

3. 查询优化

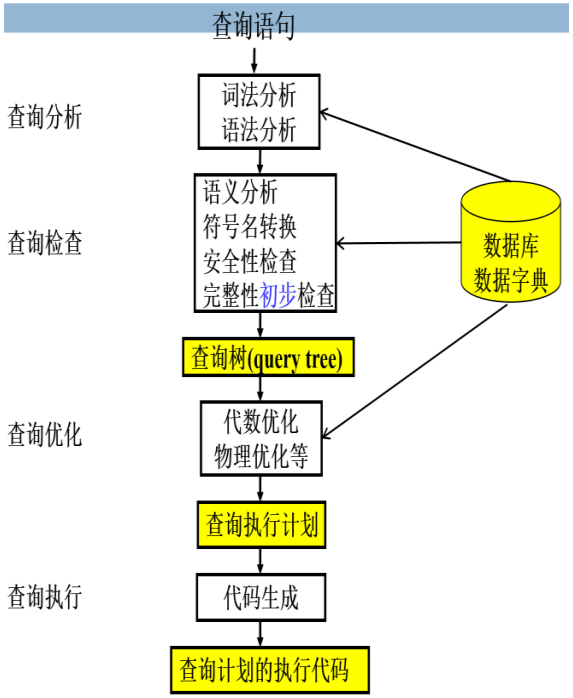
- 查询优化：选择一个高效执行的查询处理策略
- 查询优化层次分类：
  - **代数优化**：指关系代数表达式的优化
  - **物理优化**：指存取路径和底层操作算法的选择
- 查询优化方法选择的依据：
  - 基于规则(rule based)
  - 基于代价(cost based)
  - 基于语义(semantic based)

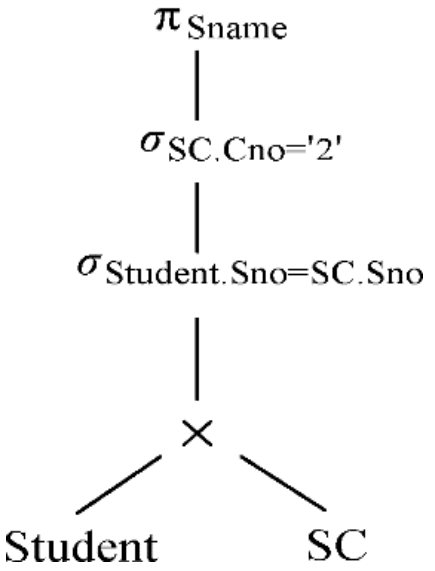
实际操作中，这些层次和方法是综合使用的。



4. 查询执行

- 依据优化器得到的执行策略生成查询计划
- 代码生成器(code generator)生成执行查询计划的代码
- 两种执行方法
  - 自顶向下
  - 自底向上





9.1 关系数据库系统的查询处理

- **9.1.1 查询处理步骤**
- **9.1.2 实现查询操作的算法示例**

9.1.2 实现查询操作的算法示例

一、 选择操作的实现

二、 连接操作的实现

一、 选择操作的实现

- ［例9.1］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；

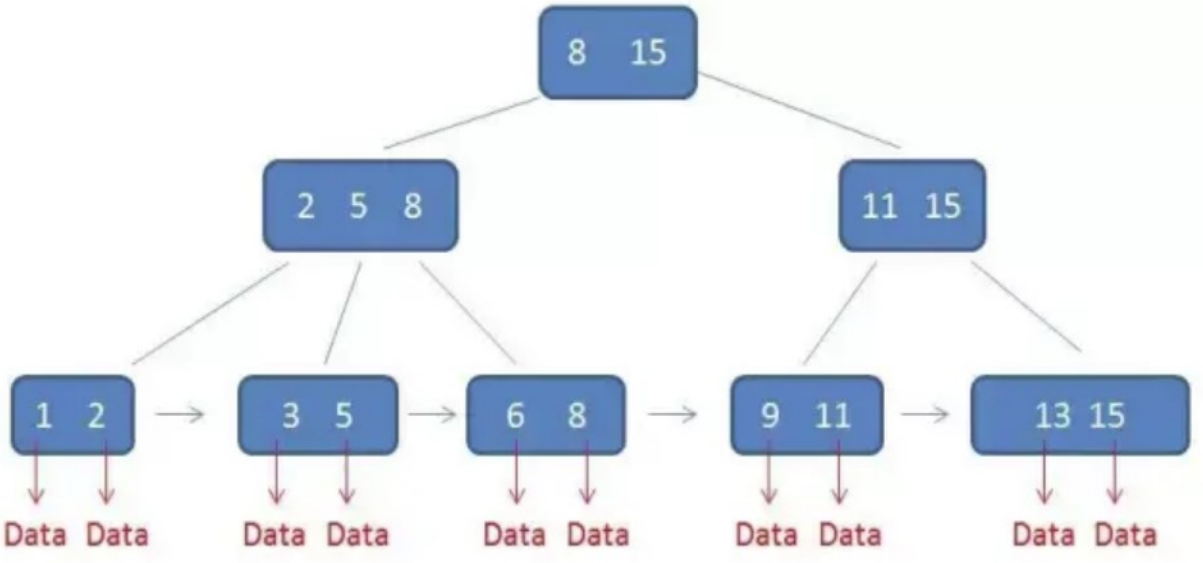
C4：Sdept＝'CS' AND Sage>20；

选择操作的实现（续）

- 选择操作典型实现方法：
  - 1. 简单的全表扫描方法
    - 对查询的基本表顺序扫描，逐一检查每个元组是否满足选择条件，把满足条件的元组作为结果输出
    - 适合小表，不适合大表
  - [例9.1- C1] Select * from student
    - Where 无条件
    - 假设可以使用的内存为M块，全表扫描算法思想：
      - (1) 按照物理次序读Student的M块到内存
      - (2) 检查内存的每个元组t，如果满足选择条件，则输出t
      - (3) 如果Student还有其他块未被处理，重复(1)和(2)

选择操作的实现（续）

- **选择操作典型实现方法：**
  - **2. 索引(或散列)扫描方法**
    - 适合选择条件中的属性上有索引(例如B+树索引或Hash索引)
    - 通过索引先找到满足条件的元组主码或元组指针，再通过元组指针直接在查询的基本表中找到元组



选择操作的实现（续）

- 索引扫描方法

  [例9.1] Select * from student where Sno='201215121';

- ［例9.1-C2］ Sno上有索引(或Sno是散列码)
  - 使用索引(或散列)得到Sno为‘200215121’元组的指针
  - 通过元组指针在student表中检索到该学生

[例9.1] Select * from student where Sage>20；

- ［例9.1-C3］ Sage>20，并且Sage 上有B+树索引
  - 使用B+树索引找到Sage＝20的索引项，以此为入口点在B+树的顺序集上得到Sage>20的所有元组指针
  - 通过这些元组指针到student表中检索到所有年龄大于20的学生

选择操作的实现（续）

- 索引扫描方法

[例9.1] SELECT * FROM Student WHERE Sdept='CS' AND Sage>20;

- ［例9.1-C4］ 如果Sdept和Sage上都有索引：
  - 算法一：分别用上面两种方法分别找到Sdept＝‘CS’的一组元组指针和Sage>20的另一组元组指针
    - 求这2组指针的交集
    - 到student表中检索
    - 得到计算机系年龄大于20的学生
  - 算法二：找到Sdept＝‘CS’的一组元组指针，
    - 通过这些元组指针到student表中检索
    - 对得到的元组检查另一些选择条件(如Sage>20)是否满足
    - 把满足条件的元组作为结果输出。

一、 选择操作的实现

- ［例9.1］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； 全表扫描

C2：Sno＝‘200215121’； 索引扫描

C3：Sage>20； 索引扫描，或全表扫描

C4：Sdept＝‘CS’ AND Sage>20； 综合考虑

二、 连接操作的实现

- 连接操作是查询处理中最耗时的操作之一
- 本节只讨论等值连接(或自然连接)最常用的实现算法

- ［例9.2］ SELECT *
  FROM Student，SC
  WHERE Student.Sno = SC.Sno；

连接操作的实现（续）

- **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**

连接操作的实现（续）

1. 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

［例9.2］SELECT *
FROM Student，SC
WHERE Student.Sno=SC.Sno；

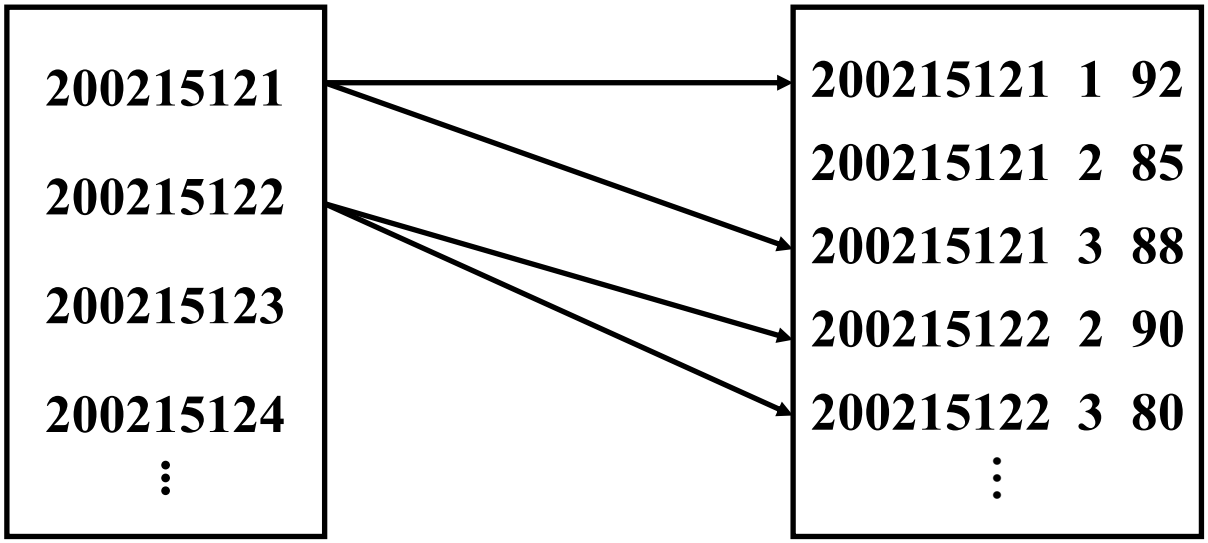
连接操作的实现（续）

2. 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
- 排序－合并连接方法的步骤：
  - Sort：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
  - Merge: 取Student表中第一个Sno，依次扫描SC表中具有相同Sno的元组
  - 当扫描到Sno不相同的第一个SC元组时，返回Student表扫描它的下一个元组，再扫描SC表中具有相同Sno的元组，把它们连接起来
  - 重复上述步骤直到Student 表扫描完

［例9.2］SELECT *
FROM Student，SC
WHERE Student.Sno=SC.Sno；

连接操作的实现（续）



排序-合并连接方法示意图

连接操作的实现（续）

- 说明
  - Student表和SC表都只要扫描一遍
  - 如果2个表原来无序，执行时间要加上对两个表的排序时间
  - 对于2个大表，先排序后使用排序-合并（sort-merge join）方法执行连接，总的时间一般仍会大大减少
  - 排序方法，参考《数据结构》

连接操作的实现（续）

3. 索引连接(index join)方法

- **步骤：**
  - **① 在SC表上建立属性Sno的索引，如果原来没有该索引**
  - **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

```
［例9.2］SELECT *
        FROM Student，SC
        WHERE Student.Sno=SC.Sno；
```

连接操作的实现（续）

4. 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(building phase)：
    - 对包含较少元组的表(比如Student)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) 或 连接阶段(join phase)
    - 对另一个表(SC)进行一遍处理
    - 把SC的元组散列到适当的hash桶中
    - 把元组与桶中所有来自Student并与之相匹配的元组连接起来

```
［例9.2］SELECT *
        FROM Student，SC
        WHERE Student.Sno=SC.Sno;
```

连接操作的实现（续）

- 上面hash join算法前提：假设两个表中较小的表在第一阶段后可以完全放入内存的hash桶中

- 以上的算法思想可以推广到更加一般的多个表的连接算法上

第九章 关系系统及其查询优化

9.2 关系数据库系统的查询优化

- 查询优化在关系数据库系统中有着非常重要的地位
- 关系查询优化是影响**RDBMS**性能的关键因素
- 由于关系表达式的语义级别很高，使关系系统可以从关系表达式中分析查询语义，提供了执行查询优化的可能性

9.2 关系数据库系统的查询优化

- **9.2.1 查询优化概述**
- **9.2.2 一个实例**

查询优化概述（续）

- 关系系统
  - 减轻了用户选择存取路径的负担
  - 用户提出“干什么”，不必指出“怎么干”
- 非关系系统
  - 用户使用过程化的语言表达查询要求，执行何种记录级的操作，以及操作的序列是由用户来决定的
  - 用户必须了解存取路径，系统要提供用户选择存取路径的手段，查询效率由用户的存取策略决定
  - 如果用户做了不当的选择，系统是无法对此加以改进的

查询优化概述（续）

- 查询优化的优点
  - 用户不必考虑如何最好地表达查询以获得较好的效率
  - 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（元组数、索引等），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。

查询优化概述（续）

- (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

查询优化概述（续）

- 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)
      - 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

查询优化概述（续）

- 查询优化的总目标：
  - 选择有效的策略
  - 求得给定关系表达式的值
  - 使得查询代价最小(实际上是较小)
  - 实际上：找较优，避免最差

9.2 关系数据库系统的查询优化

- **9.2.1 查询优化概述**
- **9.2.2 一个实例**

9.2.2 一个实例

- 一个关系查询可以对应不同的执行方案，其效率可能相差非常大

[例9.3] 求选修了2号课程的学生姓名。用SQL表达：

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

- 假设：学生-课程数据库中
  - 有1000个学生(Student)记录，10000个选课记录(SC)
  - 其中，选修2号课程的选课记录为50个
  - 每秒读写内存块数量为20块

一个实例（续）

- 系统可以用多种等价的关系代数表达式来完成这一查询

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

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

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

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

一个实例（续）

一、第一种情况

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

- 1. 计算广义笛卡尔积
- 把Student和SC的每个元组连接起来的做法：
  - 1.在内存中尽可能多地读入某个表(如Student表)的若干块，留出一块读另一个表(如SC表)的元组
  - 2.把SC中的每个元组和Student中每个元组连接，连接后的元组装满一块后就写到中间文件上
  - 3.再读入若干块Student元组，读入一块SC元组
  - 重复上述处理过程，直到把Student表处理完

一个实例（续）

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

- Student表有1000个学生，SC表有10000个选课记录
- 假设：一个块能装10个Student元组或100个SC元组，在内存中存放5块Student元组和1块SC元组，则读取总块数为

$$\frac{1000}{10}+\frac{1000}{10\times 5}\times\frac{10000}{100}=100+20\times 100=2100\text{块}$$

  - 其中，读Student表100块。读SC表20遍，每遍100块。
  - 读取Student和SC，需读取2100块。若每秒读写20块，则总计需要花105s（2100/20）
  - 执行笛卡尔积连接后的元组数：$10^3\times 10^4=10^7$。设每块能装10个元组，则$10^7/10=10^6$块（中间元组），写出这些块要用$10^6/20=5\times 10^4$s

一个实例（续）

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

- 2. 作选择操作
  - 依次读入连接后的元组，按照选择条件选取满足要求的记录，读入10^6块
  - 假定内存处理时间忽略。读取中间文件花费的时间(同写中间文件一样)， 需 5×10^4s
  - 假设满足条件的元组仅50个，均可放在内存
- 3. 作投影操作
  - 把第2步的结果在Sname上作投影输出，得到最终结果
  - 所有内存处理时间均忽略不计

❖ 执行查询的总读写数据块=$2100+10^6+10^6$

❖ 执行查询的总时间 ≈ $105+2\times5\times10^4\approx10^5$s

一个实例（续）

- 二、 第二种情况

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

1. 计算自然连接
   - 执行自然连接，读取Student和SC表的策略不变，总的读取块数仍为2100块，花费105 s
   - 自然连接的结果比第一种情况大大减少，为10^4个(中间元组)
   - 写出$10^4/10 = 10^3$块，写元组时间为$10^3/20 = 50$s，为第一种情况的千分之一 （每秒20块，每块10个元组）
2. 执行选择运算：读取中间文件块10^3块，花费时间也为50s
3. 执行投影：把第2步结果投影输出。

- 总的读写数据块 = $2100+10^3+10^3$块
- 总的执行时间≈105+50+50≈205s
- 执行代价大约是第一种情况的488分之一

一个实例（续）

- 三、 第三种情况

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

  - 1. 先对SC表作选择运算，只需读一遍SC表，存取100块，花费时间为5s。因满足条件的元组仅50个，不必使用中间文件
  - 2. 读取Student表，把读入的Student元组和内存中的SC元组作连接。只需读一遍Student表共100块，花费时间为 5s。
    - （50个SC元组只需要少于1块内存）
  - 3. 把连接结果投影输出
- 总读取块= 100 + 100 = 200块
- 总的执行时间≈ 5+5 ≈ 10s
- 执行代价约是第一种的万分之一，是第二种的20分之一

一个实例（续）

- 对于第三种情况：$Q_3=\pi_{Sname}$ (Student ⋈ $\sigma_{SC.Cno='2'}$(SC))
- 假如SC表的Cno字段上有索引
  - 第一步：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

一个实例（续）

- 把代数表达式Q_1变换为Q_2、 Q_3，

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

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

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

  - 代数优化：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
  - 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
    - 物理优化：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