

第9章 关系查询处理与查询优化

本章目标

- 完成本章的学习，你应该能够
 - 理解RDBMS查询处理的步骤及每个步骤包含的内容
 - 理解选择操作中的简单全表扫描算法和索引扫描算法的实现过程
 - 理解连接操作中的嵌套循环算法、排序-合并算法、索引连接算法和hash连接算法的实现过程
 - 理解查询优化器在RDBMS中的重要性
 - 熟练掌握如何将查询语句转换为查询树以及对查询树进行查询优化的方法
 - 掌握选择操作的启发式规则和连接操作的启发式规则
 - 掌握基于代价估算的优化方法

主要概念

- 查询处理(Query processing)
 - 指RDBMS执行查询语句的过程
 - 任务：把用户提交给RDBMS的查询语句转换为高效的查询执行计划
- 代数优化
 - 也称逻辑优化
 - 指关系代数表达式的优化
- 物理优化
 - 也称非代数优化
 - 指存取路径和底层操作算法的选择进行的优化

大纲

- **关系数据库系统的查询处理**
- 关系数据库系统的查询优化
- 代数优化
- 物理优化
- 查询计划的执行
- 本章小结

Recap

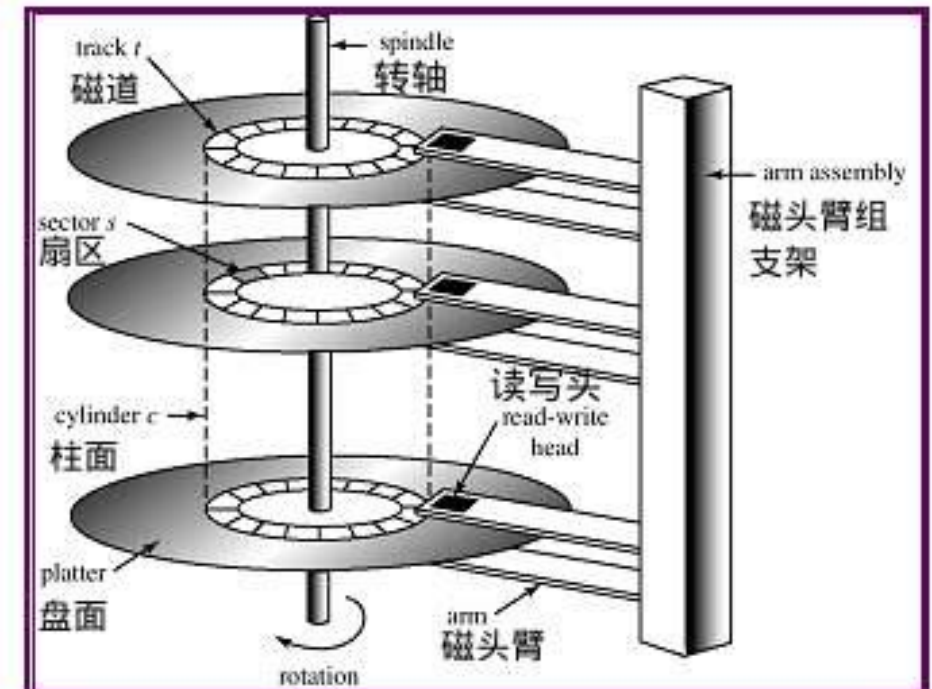
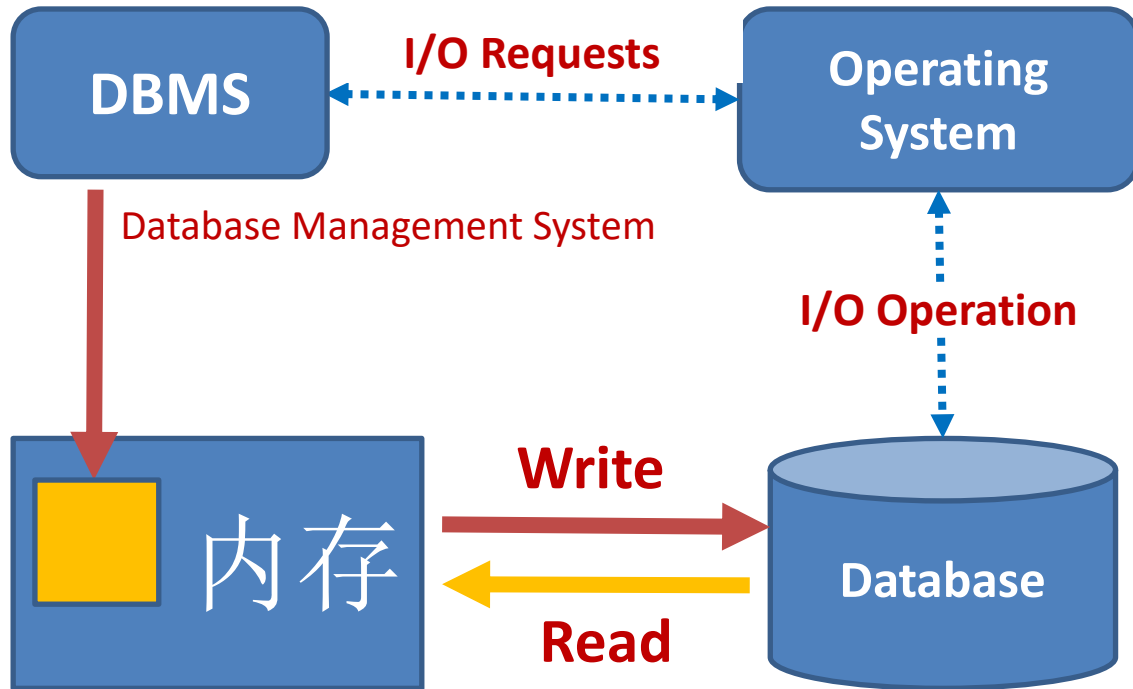
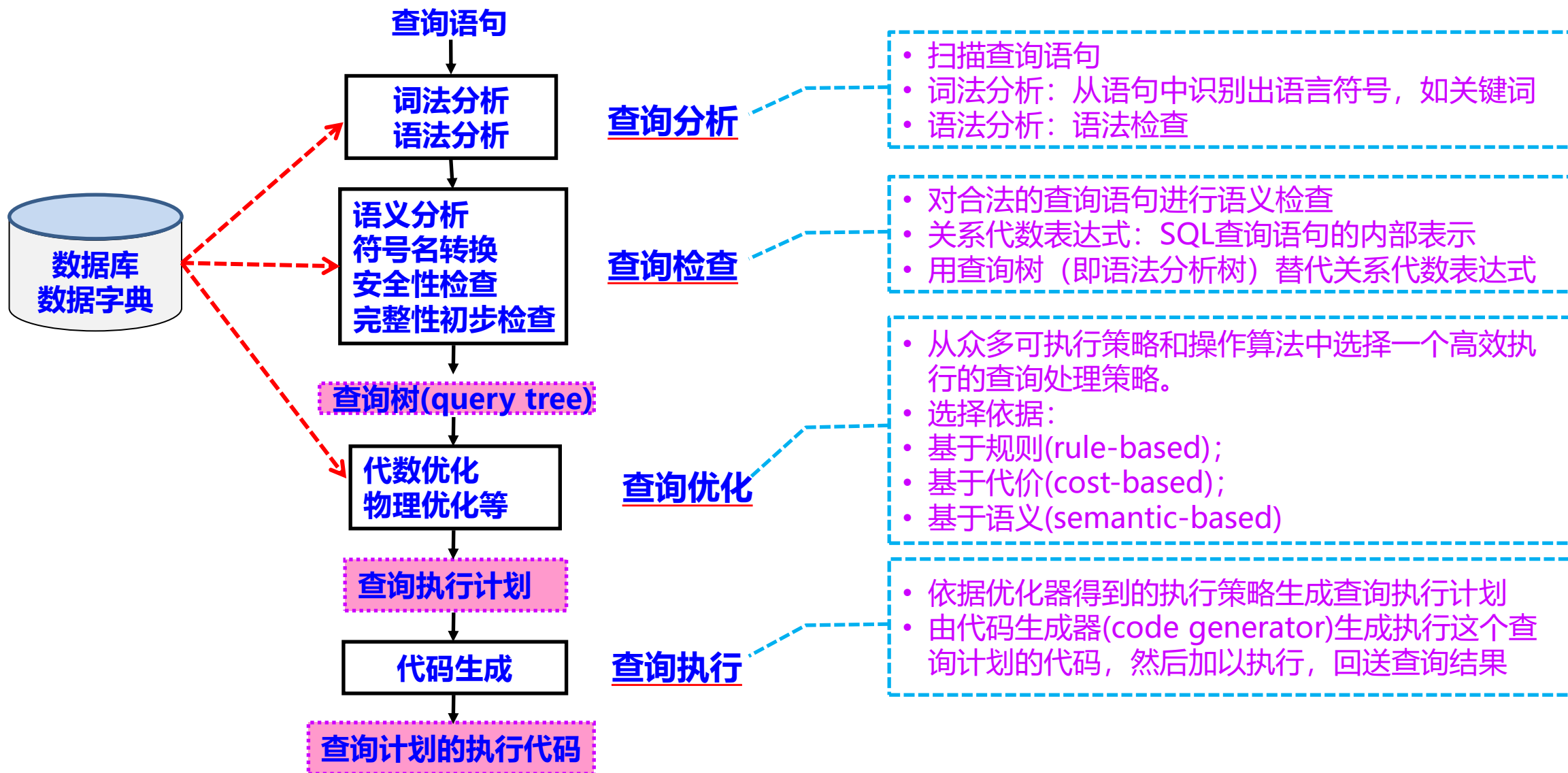


Fig. Conventional Architecture of DBMS

关系数据库系统的查询处理



openGauss的查询处理与优化

- 补充

Oracle 10g的查询计划

■ 获取查询计划

- 获取Oracle优化器在执行SELECT、UPDATE、DELETE和INSERT时选择的查询计划.

EXPLAIN PLAN FOR <SELECT...> | <UPDATE...> | <DELETE...> | <INSERT...>

■ 查看执行计划

SELECT * FROM table(DBMS_XPLAN.display);

■ 优化器的缺省模式

SHOW PARAMETERS OPTIMIZER_MODE



具有DBA权限的
用户才能查看

```
SQL> show parameters optimizer_mode
```

NAME	TYPE	VALUE
optimizer_mode	string	ALL_ROWS

参考资料: https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i26093


```
SQL> explain plan for select deptno, dname from dept where LOC='BOSTON';
```

已解释。

```
SQL> select * from table(dbms_xplan.display);
```

PLAN_TABLE_OUTPUT

Plan hash value: 3383998547

| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |

| 0 | SELECT STATEMENT | | 1 | 20 | 3 (0)| 00:00:01 |
|* 1 | TABLE ACCESS FULL| DEPT | 1 | 20 | 3 (0)| 00:00:01 |

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT

1 - filter("LOC"='BOSTON')

已选择13行。

```
SQL> select * from table(dbms_xplan.display);
```

```
PLAN_TABLE_OUTPUT
```

```
-----  
Plan hash value: 2949544139  
-----
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		1	10	1 (0)	00:00:01
1	TABLE ACCESS BY INDEX ROWID	EMP	1	10	1 (0)	00:00:01
* 2	INDEX UNIQUE SCAN	PK_EMP	1		0 (0)	00:00:01

```
-----  
Predicate Information (identified by operation id):  
-----
```

```
PLAN_TABLE_OUTPUT
```

```
-----  
2 - access("EMPNO">=10)
```

```
已选择14行。
```

实现查询操作的算法示例

■ 选择操作的实现算法

– 全表扫描方法 (Full Table Scan)

- 用于规模小的表, 简单有效
- 用于大表, 当选择率较低时, 效率很低

– 索引扫描方法 (Index Scan)

- 当选择率较低时, 性能优于全表扫描
- 当选择率较高或查找的元组均匀分布时, 性能低于全表扫描

假设可使用的内存为M块, 全表扫描算法:

- 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

[例9.1-C2] 假设Sno上有索引, 索引扫描算法:

- 使用索引得到Sno为'201215121'元组的指针
- 通过元组指针在Student表中检索到该学生

[例9.1-C3] 假设Sage上有B+树索引, 索引扫描算法:

- 使用B+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
- 再通过元组指针在Student表中检索到所有年龄大于20的学生

[例9.1-C4] 假设Sdept和Sage上都有索引:

- 算法一: 对每个条件同时采用索引算法, 分别找到相应的元组指针, 求它们的交集, 再到Student表检索
- 算法二: 先找到满足第一个条件的元组指针, 再在Student表中检索到相应的元组, 检查这些元组是否满足第二个条件, 若满足, 则输出结果

[例9.1] SELECT * FROM Student WHERE <条件表达式>

- C1: 无条件;
- C2: Sno='201215121';
- C3: Sage>20;
- C4: Sdept='CS' AND Sage>20;

■ 连接操作的实现算法

- 连接操作是查询处理中最常用和最耗时的操作之一
- 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;`

- 嵌套循环算法(nested loop join)
- 排序-合并算法(sort-merge join 或merge join)
- 索引连接(index join)算法
- Hash Join算法
 - 1.划分阶段, 即创建hash表: 把小表中的元组按hash函数分散到hash表的桶中
 - 2.试探阶段, 即连接阶段: 对另一张表按相同的hash函数处理, 并放到适当的hash桶, 然后把匹配的元组连接起来
 - 假设小表的hash值在划分阶段后可以完全放入内存的hash桶中

大纲

- 关系数据库系统的查询处理
- **关系数据库系统的查询优化**
- 代数优化
- 物理优化
- 查询计划的执行
- 本章小结

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
- 本节内容
 - 查询优化概述
 - 一个实例

查询优化概述

■ 关系系统的查询优化

- 是RDBMS实现的关键技术又是关系系统的优点所在
- 减轻了用户选择存取路径的负担

■ 非关系系统的查询优化

- 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

■ 查询优化的优点

- 用户不必考虑如何最好地表达查询以获得较高的效率
- 系统可以比用户程序的“优化”做得更好
 - 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。
 - 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。
 - 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。
 - 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
- 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（开销通常以块数为衡量单位）
 - 集中式数据库开销：总代价=I/O代价+CPU代价+内存代价
 - 分布式数据库开销：总代价=I/O代价+CPU代价+内存代价+通信代价

■ 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，使得查询代价较小

一个实例

■ [例9.3] 求选修了2号课程的学生姓名

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

- 假定学生-课程数据库中有1000个学生记录，10000个选课记录，选修2号课程的选课记录为50个
- 实现上述查询的等价关系代数表达式有：

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

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

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

情形一： $Q_1 = \Pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='2'}(Student \times SC))$

1. 计算广义笛卡尔积

— 算法

- 在内存中尽可能多地装入某个表(如Student表)的若干块，留出一块存放另一个表(如SC表)的元组。
- 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
- 从SC中读入一块和内存中的Student元组连接，直到SC表处理完。
- 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

— 代价计算

- 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 $\frac{1000}{10} + \frac{1000}{10 \times 5} \times \frac{10000}{100} = 100 + 20 \times 100 = 2100$ 块
- 连接后的元组数为 $10^3 \times 10^4 = 10^7$ 。设每块能装10个元组，则写出 10^6 块

情形一： $Q_1 = \Pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='2'}(Student \times SC))$

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
- 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)需读入106块。
- 若满足条件的元组假设仅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

■ 情形一下执行查询的总读写数据块 = $2100 + 10^6 + 10^6$

情形二： $Q_2 = \Pi_{Sname}(\sigma_{SC.Cno='2'}(Student \bowtie SC))$

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
- 自然连接的结果比第一种情况大大减少，为 10^4 个元组
- 写出数据块 = 10^3 块

2. 读取中间文件块，执行选择运算，读取的数据块 = 10^3 块

3. 把第2步结果投影输出

- 情形二下执行查询的总读写数据块 = $2100 + 10^3 + 10^3$ ，其执行代价大约是情形一的 $\frac{1}{488}$ 分之一

情形三： $Q_3 = \Pi_{Sname}(Student \bowtie \sigma_{SC.Cno='2'}(SC))$

1. 计算自然连接先对SC表作选择运算，只需读一遍SC表，存取100块，因为满足条件的元组仅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。也只需读一遍Student表共100块
3. 把连接结果投影输出

- 情形三下执行查询的总读写数据块=100+100，其执行代价是情形一的万分之一，情形二的二十分之一

■ 分析与总结

- 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'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，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
- 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，这就是代数优化
- 在Q₃中，SC表的选择操作算法有全表扫描或索引扫描，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，采用索引连接代价也较小，这就是物理优化

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的基本方法和思路

大纲

- 关系数据库系统的查询处理
- 关系数据库系统的查询优化
- **代数优化**
- 物理优化
- 查询计划的执行
- 本章小结

代数优化

- SQL语句经过查询分析、查询检查后变换为查询树，查询树是关系代数表达式的内部表示。
- 代数优化
 - 通过对关系代数表达式的等价变换来提高查询效率
 - 关系代数表达式的等价是指用相同的关系替代两个表达式中相应的关系所得到的结果是相同的
- 关系代数等价变换规则
 - 11个等价变换规则

■ 查询树的启发式优化

– 典型的启发式规则(heuristic rule)有

- ① 选择运算应尽可能先做：在优化策略中这是最重要、最基本的一条
- ② 把投影运算和选择运算同时进行：如有若干投影和选择运算，并且它们都对同一个关系操作，则可以在扫描此关系的同时完成所有的这些运算以避免重复扫描关系
- ③ 把投影同其前或其后的双目运算结合起来，没有必要为了去掉某些字段而扫描一遍关系
- ④ 把某些选择同在它前面要执行的笛卡尔积结合起来成为一个连接运算，连接特别是等值连接运算要比同样关系上的笛卡尔积省很多时间
- ⑤ 找出公共子表达式：如果这种重复出现的子表达式的结果不是很大的关系，并且从外存中读入这个关系比计算该子表达式的时间少得多，则先计算一次公共子表达式并把结果写入中间文件是合算的。当查询的是视图时，定义视图的表达式就是公共子表达式的情况

■ 查询树的启发式优化算法

算法：关系表达式的优化

输入：一个关系表达式的查询树

输出：优化的查询树

步骤：

1. 利用等价变换规则4，把形如 $\sigma_{F_1 \wedge F_2 \wedge \dots \wedge F_n}(E)$ 变换为 $\sigma_{F_1}(\sigma_{F_2}(\dots(\sigma_{F_n}(E))\dots))$
 2. 对每一个选择，利用等价变换规则4~9尽可能把它移到树的叶端
 3. 对每一个投影，利用等价变换规则3、5、10、11中的一般形式尽可能把它移向树的叶端
 4. 利用等价变换规则3~5，把选择和投影的串接合并成单个选择、单个投影或一个选择后跟一个投影，使多个选择或投影能同时执行，或在一次扫描中全部完成
 5. 把上述得到的语法树的内节点分组。每一双目运算（ \times ， \bowtie ， \cup ， $-$ ）和它所有的直接祖先为一组（ σ ， Π ）。如果其后代直到叶子全是单目运算，则也将它们并入该组，但当双目运算是笛卡尔积（ \times ），且后面不是与它组成等值连接的选择时，则不能把选择与这个双目运算组成同一组。把这些单目运算单独分为一组。
-

■ [例9.4] 对[例9.3]中的SQL语句进行代数优化

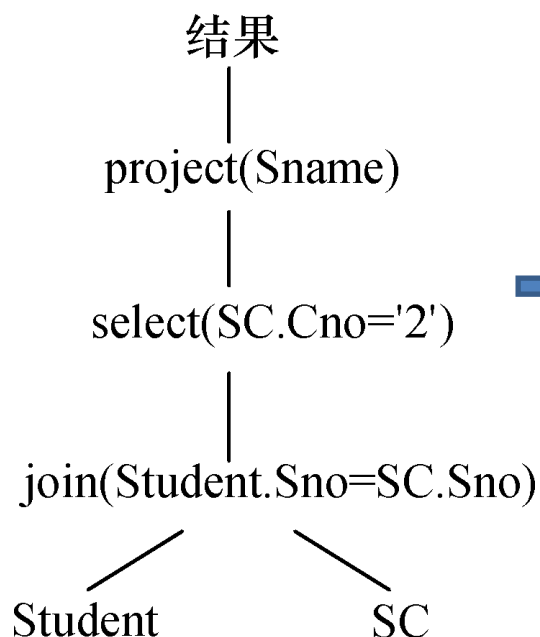
SELECT Student.Sname FROM Student,SC WHERE Student.Sno=SC.Sno AND SC.Cno='2' ;

步骤:

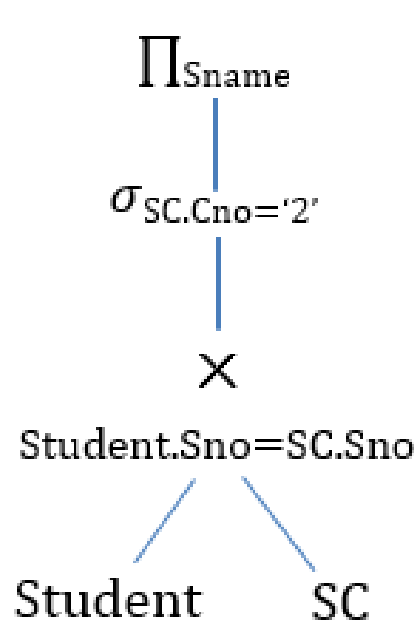
1. 将SQL语句用查询树表示

2. 转化为关系代数语法树

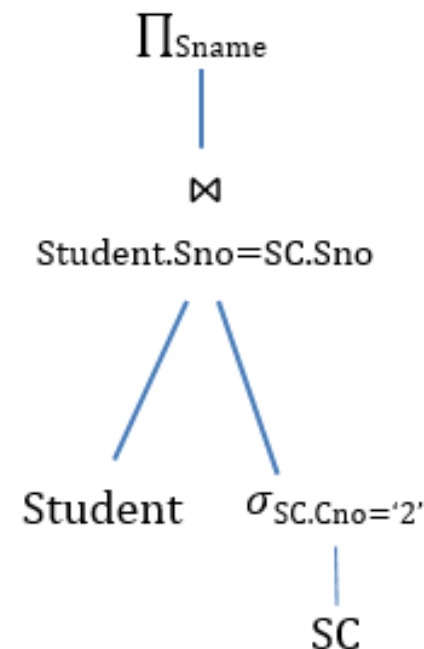
3. 对关系代数语法树优化



叶节点: 查询的输入基本表
中间节点: 关系代数运算
根节点: 关系代数的最后一个运算

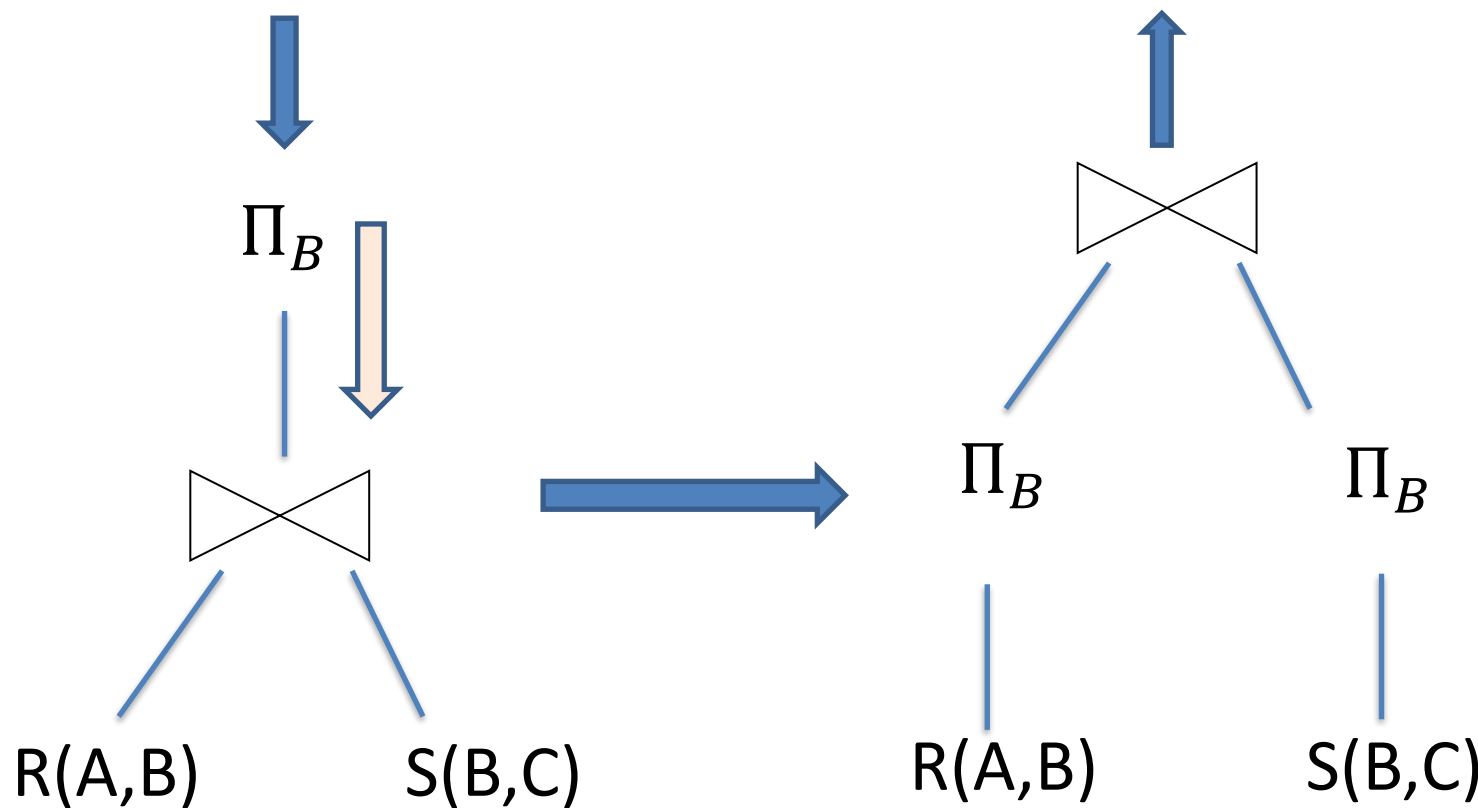


该图与教材略有不同



- [例子] 对关系代数表达式 $\Pi_B(R(A, B) \bowtie S(B, C))$ 进行查询优化

$$\Pi_B(R(A, B) \bowtie S(B, C)) \quad \Pi_B R(A, B) \bowtie \Pi_B S(B, C)$$



■ [例子]

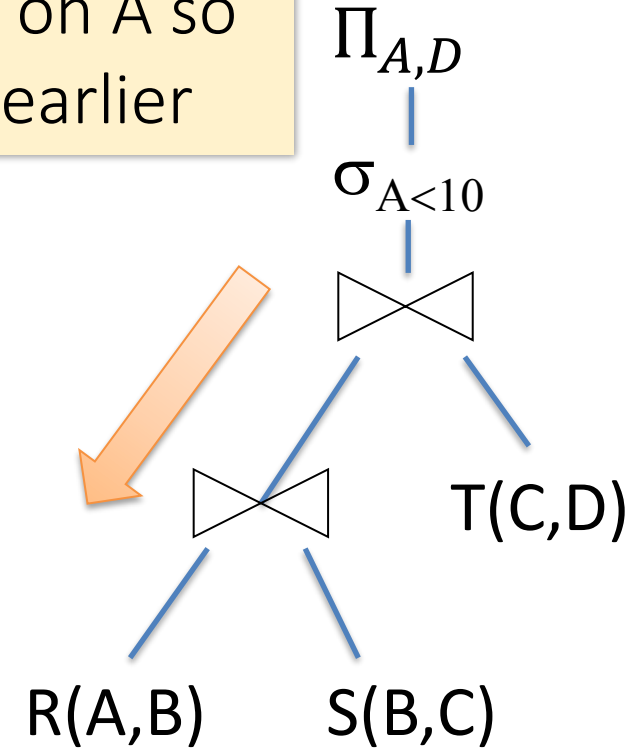
R(A,B) S(B,C) T(C,D)

```
SELECT R.A,T.D  
FROM R,S,T  
WHERE R.B = S.B  
      AND S.C = T.C  
      AND R.A < 10;
```



$$\Pi_{A,D}(\sigma_{A<10}(T \bowtie (R \bowtie S)))$$

Push down
selection on A so
it occurs earlier



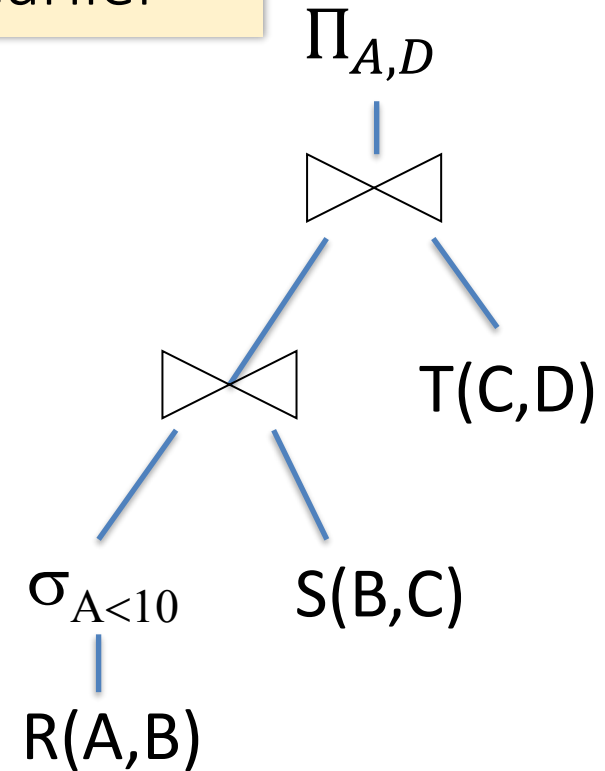
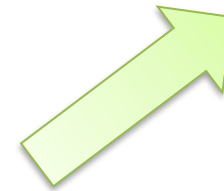
R(A,B) S(B,C) T(C,D)

SELECT R.A,T.D
FROM R,S,T
WHERE R.B = S.B
AND S.C = T.C
AND R.A < 10;



$\Pi_{A,D}(T \bowtie (\sigma_{A < 10}(R) \bowtie S))$

Push down
selection on A so
it occurs earlier



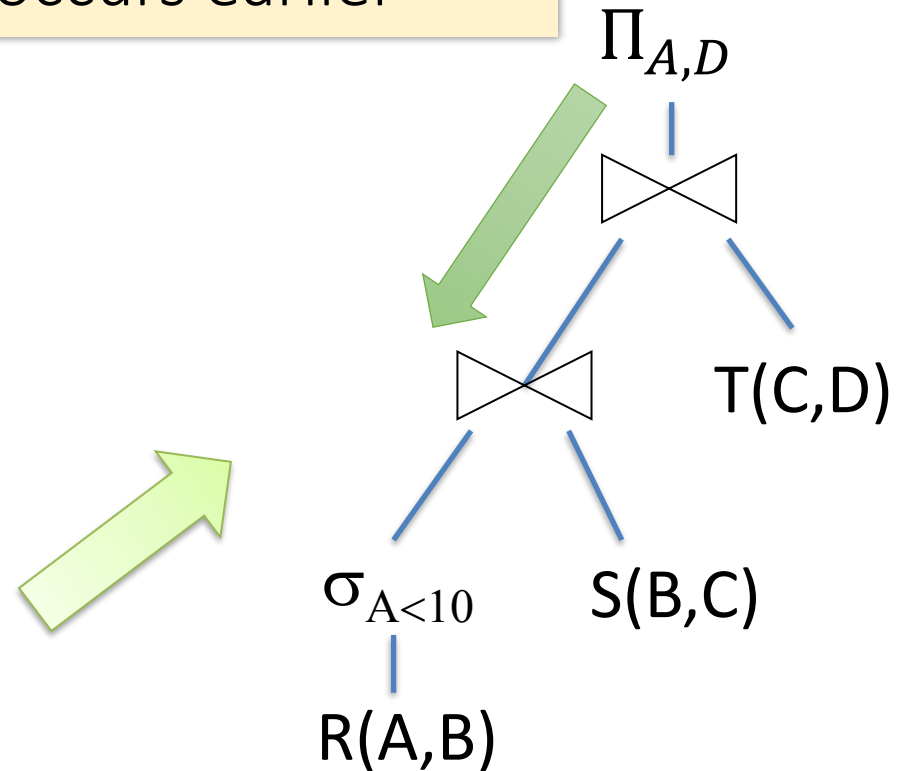
R(A,B) S(B,C) T(C,D)

SELECT R.A,T.D
FROM R,S,T
WHERE R.B = S.B
AND S.C = T.C
AND R.A < 10;



$\Pi_{A,D}(T \bowtie (\sigma_{A < 10}(R) \bowtie S))$

Push down
projection so it
occurs earlier



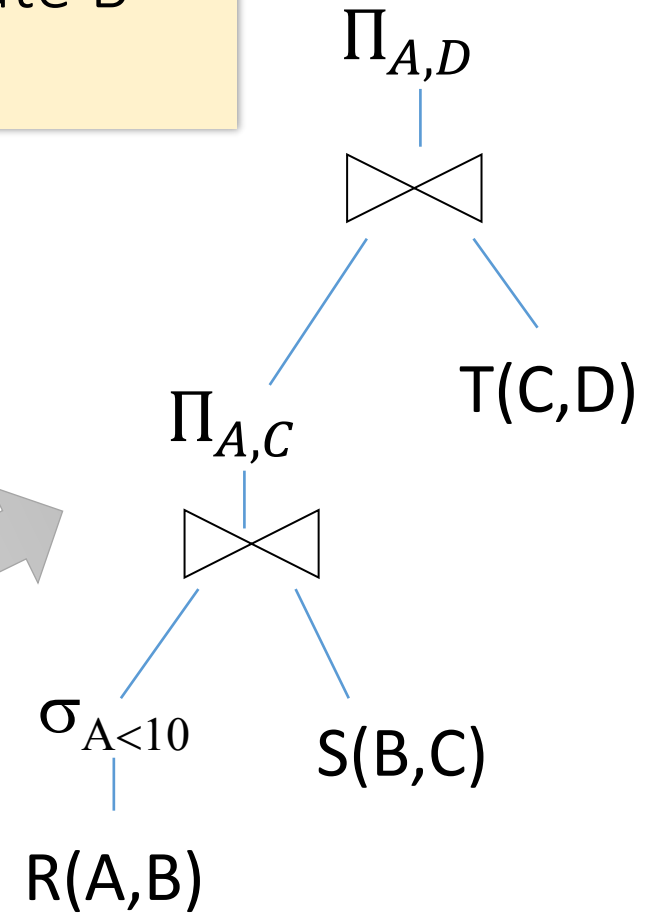
R(A,B) S(B,C) T(C,D)

SELECT R.A,T.D
FROM R,S,T
WHERE R.B = S.B
AND S.C = T.C
AND R.A < 10;



$\Pi_{A,D} \left(T \bowtie \Pi_{A,C} (\sigma_{A < 10}(R) \bowtie S) \right)$

We eliminate B
earlier!



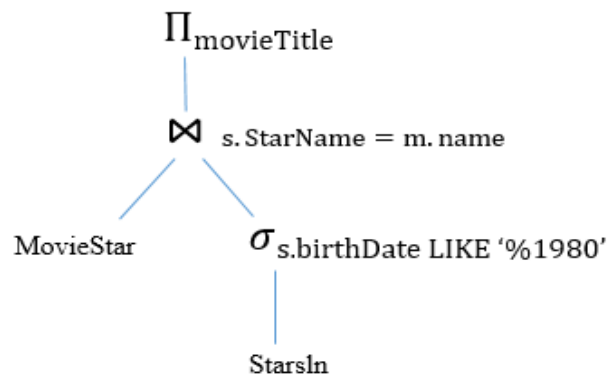
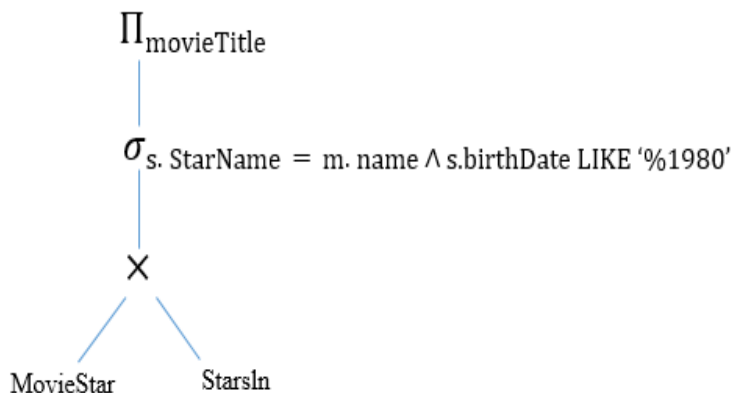
课堂练习

- 现有SQL语句如下：

```
SELECT movieTitle
FROM StarsIn s, MovieStar m
WHERE s.starName = m.name AND s.birthDate LIKE '%1980';
```

要求：

1. 画出SQL语句的初始查询树；
2. 如果该查询树不是优化的，请画出优化后的查询树



大纲

- 关系数据库系统的查询处理
- 关系数据库系统的查询优化
- 代数优化
- **物理优化**
- 查询计划的执行
- 本章小结

物理优化

- 代数优化改变查询语句中操作的次序和组合，但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，有的会相差很大，因此，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
- 典型优化方法：
 1. 基于规则的启发式优化
 - 大多数情况下都适用，但不是在每种情况下都是适用
 2. 基于代价估算的优化
 - 优化器估算不同执行策略的代价，并选出具有最小代价的执行计划
 3. 两者结合的优化方法

- 基于启发式规则的存取路径选择优化
 - 选择操作的启发式规则
 - 对于小关系，使用全表顺序扫描，即使选择列上有索引
 - 对于大关系，启发式规则有
 - (1) 对于选择条件是“主码 = 值”的查询
 - 查询结果最多是一个元组，可以选择主码索引
 - 一般的关系数据库管理系统会自动建立主码索引
 - (2) 对于选择条件是“非主属性 = 值”的查询，并且选择列 上有索引
 - 要估算查询结果的元组数目
 - 如果比例较小(<10%)可以使用索引扫描方法
 - 否则还是使用全表顺序扫描

(3) 对于选择条件是属性上的非等值查询或者范围查询，并且选择列上有索引
要估算查询结果的元组数目

- 如果比例较小($<10\%$)可以使用索引扫描方法
- 否则还是使用全表顺序扫描

(4) 对于用AND连接的合取选择条件

- 要估算查询结果的元组数目

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，可以用索引扫描方法

- 通过分别查找满足每个条件的指针，求指针的交集
- 通过索引查找满足部分条件的元组，然后在扫描这些元组时判断是否满足剩余条件

其他情况：使用全表顺序扫描

(5) 对于用OR连接的析取选择条件，一般使用全表顺序扫描

— 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-合并算法

(2) 如果一个表在连接属性上有索引

- 选用索引连接算法

(3) 如果上面2个规则都不适用，其中一个表较小

- 选用Hash join算法

(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)。

理由：

- 设连接表R与S分别占用的块数为 B_r 与 B_s
- 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 $B_r + B_r B_s / (K-1)$
- 显然应该选块数小的表作为外表

■ 基于代价的优化

- 启发式规则优化是定性的选择，比较粗糙，但实现简单且优化本身的代价较小，适合解释执行的系统。
 - 解释执行的系统其优化开销包含在查询总开销中
 - 在编译执行的系统中，一次编译优化，多次执行，查询优化和查询执行是分开执行的
- 精细复杂的基于代价的优化方法

1. 统计信息

优化器需要的统计信息存储在数据字典中

- 每个基本表的元组总数(N)、元组长度(l)、占用的块数(B)、占用的溢出块(BO)
- 每个基本表的每列，该列不同值的个数(m)、该列最大值和最小值、该列是否建立了索引、索引类型($B+$ 树、hash索引、聚集索引)。谓词条件的选择率(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，则计算每个值的选择率， f =具有该值的元组数/ N
- 对索引，索引的层数(L)、不同索引值的个数、索引的选择基数(S)、索引的叶结点数(Y)

2. 代价估算示例

(1) 全表扫描算法的代价估算公式

- 如果基本表大小为B块，全表扫描算法的代价 $\text{cost} = B$;
- 如果选择条件是“码=值”，则平均搜索代价 $\text{cost} = B/2$ 。

(2) 索引扫描算法的代价估算公式

- 如果选择条件是“码=值”，则采用该表的主索引，若为B+树，层数为L，需要存取B+树中从根结点到叶结点L块，再加上基本表中该元组所在的那一块，所以 $\text{cost} = L + 1$
- 如果选择条件涉及非码属性，若为B+树索引，选择条件是相等比较，S是索引的选择基数(有S个元组满足条件)。因为满足条件的元组可能会保存在不同的块上，所以(最坏的情况) $\text{cost} = L + S$
- 如果比较条件是 $>$ ， $> =$ ， $<$ ， $< =$ 操作，假设有一半的元组满足条件，那么就要存取一半的叶结点，并通过索引访问一半的表存储块， $\text{cost} = L + Y/2 + B/2$ 。如果可以获得更准确的选择基数，可以进一步修正Y/2与B/2

(3) 嵌套循环连接算法的代价估算公式

- 嵌套循环连接算法的代价 $\text{cost} = B_r + B_r B_s / (K - 1)$
- 如果需要把连接结果写回磁盘，则 $\text{cost} = B_r + B_r B_s / (K - 1) + (\text{Frs} * \text{Nr} * \text{Ns}) / \text{Mrs}$,
- Frs为连接选择性(join selectivity)，表示连接结果元组数的比例，
- Mrs是存放连接结果的块因子，表示每块中可以存放的结果元组数目

(4) 排序-合并连接算法的代价估算公式

- 如果连接表已经按照连接属性排好序，则 $\text{cost} = B_r + B_s + (\text{Frs} * \text{Nr} * \text{Ns}) / \text{Mrs}$
- 如果必须对文件排序，那么还需要在代价函数中加上排序的代价：对于包含B个块的文件排序的代价大约是 $(2 * B) + (2 * B * \log_2 B)$

■ 语义优化

- 查询优化器首先检测约束，如果约束不满足，则无需执行查询；否则，按之前的过程进行查询优化

大纲

- 关系数据库系统的查询处理
- 关系数据库系统的查询优化
- 代数优化
- 物理优化
- **查询计划的执行**
- 本章小结

查询计划的执行

- 查询优化完成之后，RDBMS为用户查询生成一个查询计划。
- 查询计划的执行方式
 - 自顶向下
 - 系统反复向查询计划顶端的操作符发出需要查询结果元组的请求，操作符收到请求后，试图计算下一个(几个)元组并返回这些元组。在计算时，如果操作符的输入缓冲区为空，它就会向其孩子操作符发送需求元组的请求，这种请求会一直传到叶子结点，启动叶子操作符运行，并返回其父操作符一个(几个)元组，父操作符再计算自己的输出返回给上层操作符，直到顶端操作符。重复这一过程直至处理整个关系。
 - 被动的、需求驱动的执行方式
 - 自底向上
 - 查询计划从叶节点开始执行，叶结点操作符不断产生元组并将它们放入其输出缓冲区，直到缓冲区填满为止，此时它必须等待其父操作符将元组从缓冲区中取走才能继续执行，然后其父操作符开始执行，利用下层的输入元组来产生它自己的输出元组，直到其输出缓冲区满为止。重复该过程直到产生所有的输出元组。
 - 主动的执行方式

本章小结

- 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
- 本章仅介绍了查询操作，这是RDBMS语言处理中最重要、最复杂的部分。
- 介绍了启发式代数优化、基于规则的存取路径优化和基于代价估算的优化方法。
- 对比较复杂的查询，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
- 对于RDBMS不能优化的查询需要重写查询语句，进行手工调整以优化性能

本章作业

- 教材第9章课后全部习题，即1-7题。