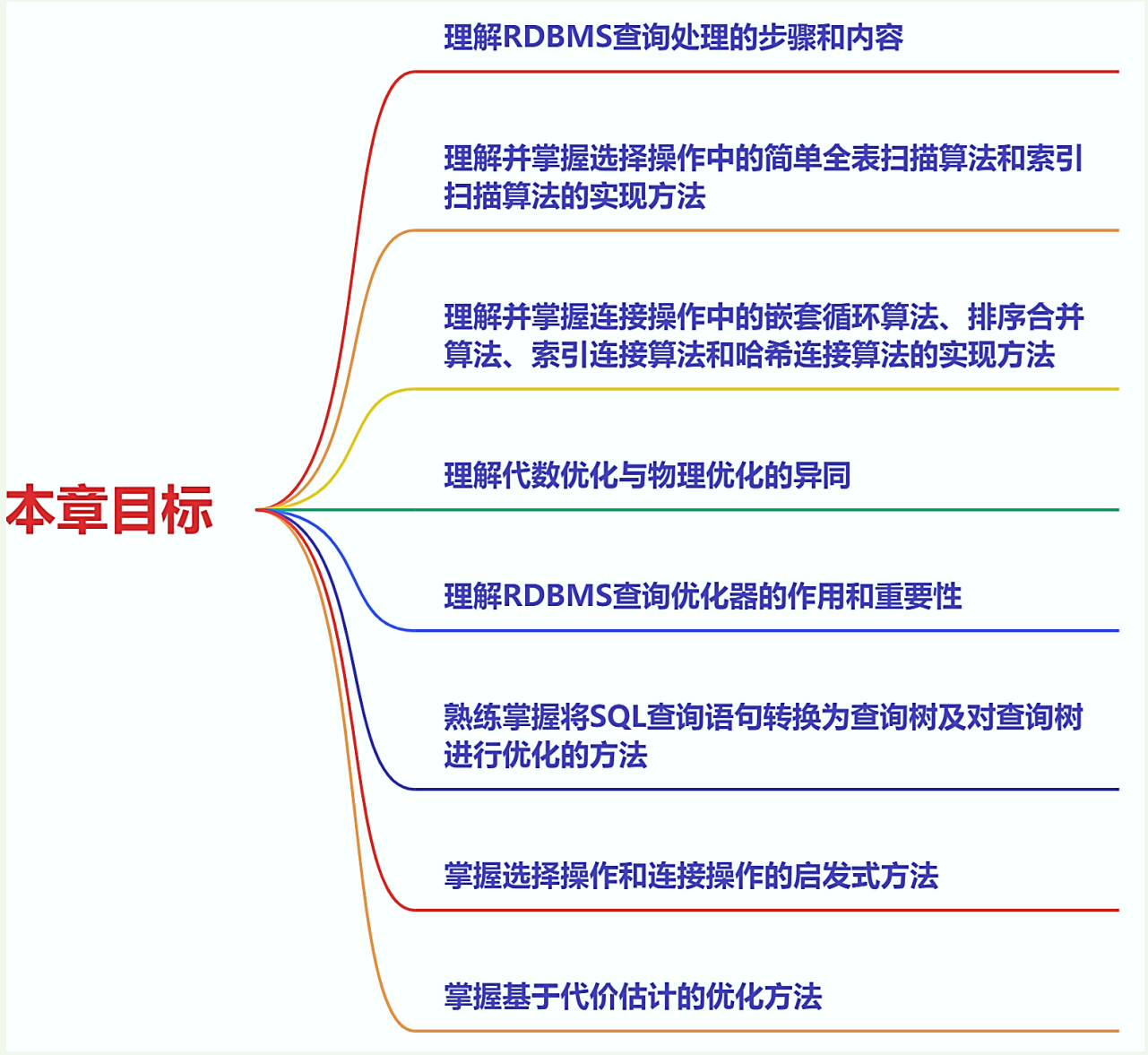
第10章 关系查询处理与查询优化

1 本章导图


本章目标
理解RDBMS查询处理的步骤和内容
理解并掌握选择操作中的简单全表扫描算法和索引扫描算法的实现方法
理解并掌握连接操作中的嵌套循环算法、排序合并算法、索引连接算法和哈希连接算法的实现方法
理解代数优化与物理优化的异同
理解RDBMS查询优化器的作用和重要性
熟练掌握将SQL查询语句转换为查询树及对查询树进行优化的方法
掌握选择操作和连接操作的启发式方法
掌握基于代价估计的优化方法

2 背景

- 存储结构和存取路径对用户透明是关系数据库的一个重要特点
  - 用户访问数据库时，只需要用SQL语句表达查询请求，具体实现过程将由RDBMS完成
- 查询处理和查询优化技术极大影响RDBMS的性能
- 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并执行
- 查询优化(Query optimization)
  - 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

3 大纲

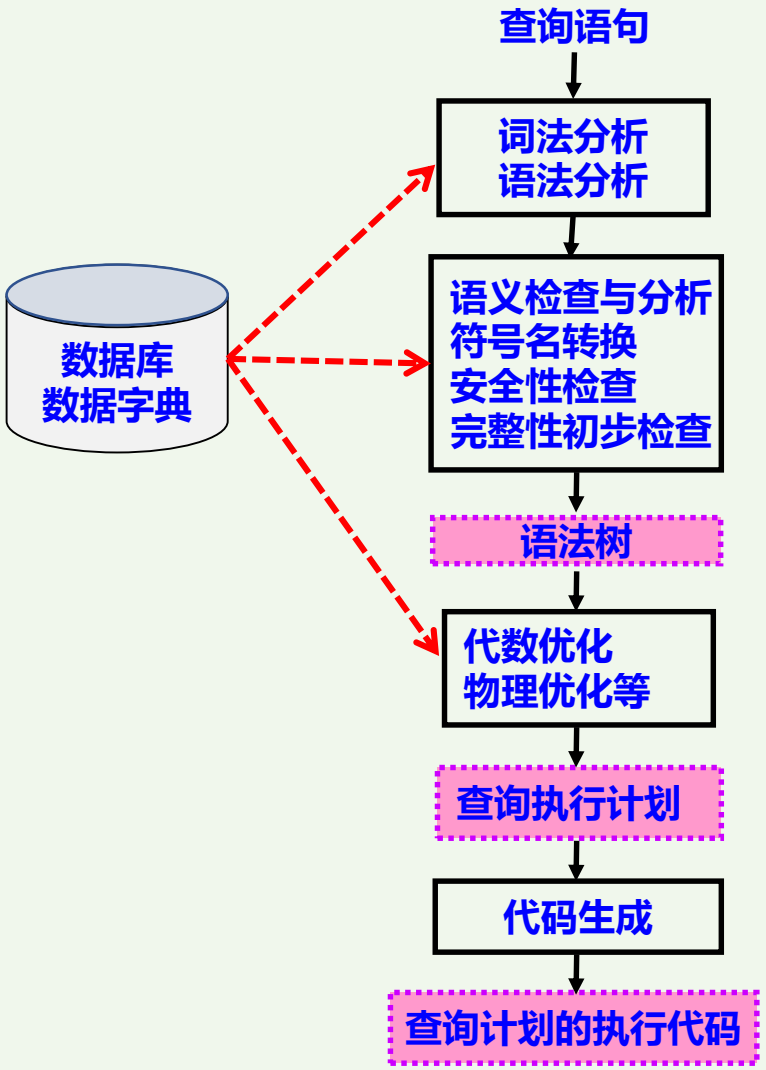
- RDBMS的查询处理
- RDBMS的查询优化
- 代数优化
- 物理优化
- 查询计划的执行
- 本章小结

4 RDBMS的查询处理

厦門大學 XIAMEN UNIVERSITY

- 本节主要内容：
  - **RDBMS的查询处理步骤（4个阶段）**
    - 查询分析
    - 查询检查
    - 查询优化
    - 查询执行
  - **实现查询操作的算法**

4.1RDBMS的查询处理步骤



查询分析

- 扫描查询语句
- 词法分析：从语句中识别出正确的语言符号
- 语法分析：语句的语法检查和是否符合SQL语法规则

查询检查

- 对合法的查询语句进行语义检查和分析
- 关系代数表达式：SQL查询语句的内部表示
- 用语法树替代表示扩展的关系代数表达式

查询优化

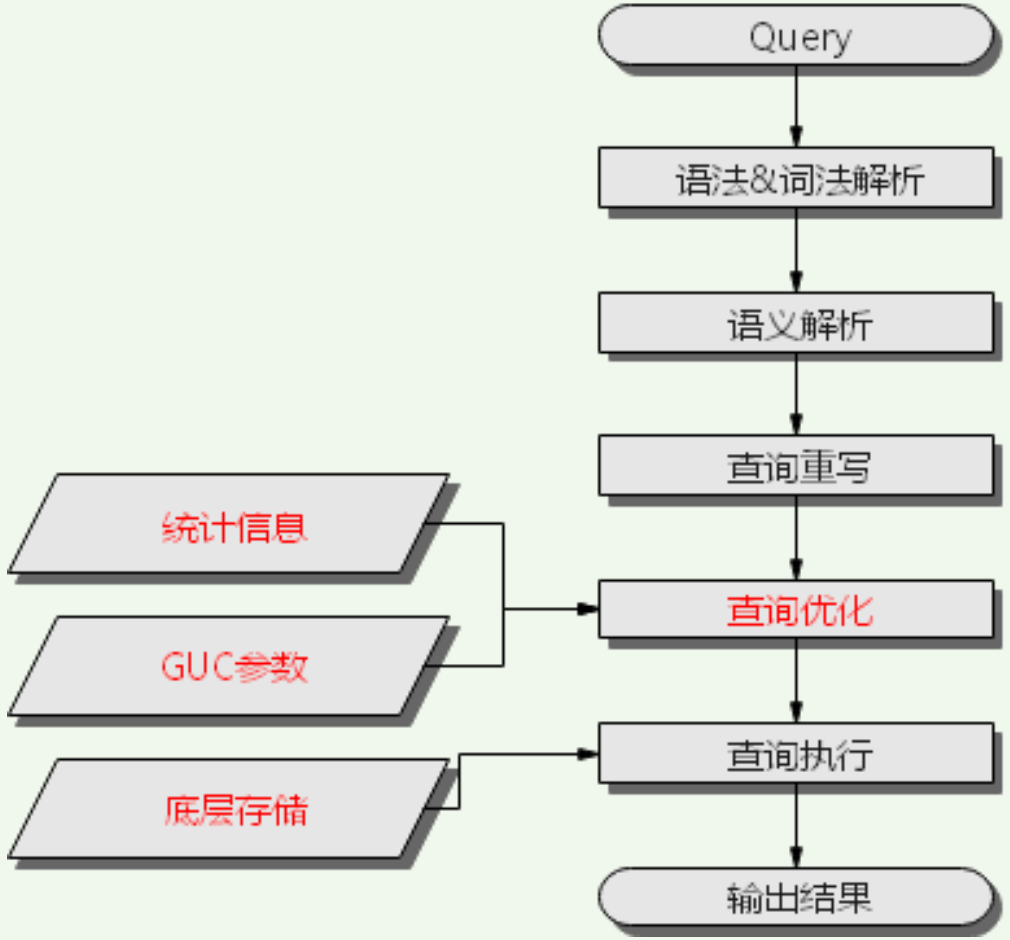
- 从众多可执行策略和操作算法中选择一个高效执行的查询处理策略。
- 优化方法的选择依据：
- 基于规则(rule-based)；
- 基于代价(cost-based)；
- 基于语义(semantic-based)

查询执行

- 依据优化器得到的执行策略生成查询执行计划
- 由代码生成器(code generator)生成执行这个查询计划的代码，然后加以执行，回送查询结果

图10.1 查询处理步骤

openGauss的查询处理与优化



- openGauss的查询处理与优化参见：

https://www.opengauss.org/zh/docs/3.1.0/docs/Developerguide/Query%E6%89%A7%E8%A1%8C%E6%B5%81%E7%A8%8B.html

- 墨天轮材料：

https://www.modb.pro/db/160703

- GUC: Global unified configuration

4 Oracle 10g的查询计划

厦門大學 XIAMEN UNIVERSITY

- 获取查询计划：

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

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

- 查看执行计划：

  SELECT * FROM table(DBMS_XPLAN.display);

- 优化器的缺省模式：

  show parameter optimizer_mode

```
SQL> show parameters optimizer_mode

NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------
optimizer_mode                       string      ALL_ROWS
```

具有DBA权限用户才能查看

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

4 Oracle 10g的查询计划(cont.)

```
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。
```

4 Oracle 10g的查询计划(cont.)

```
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。
```

4.2 RDBMS实现查询操作的算法

- 本小节简要介绍查询操作中实现选择操作和连接操作的算法思想。
  - **选择操作的实现**
  - **连接操作的实现**

4.2.1选择操作的实现

- **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)
    - 一般情况下当选择率较低时，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

[例10.1]
SELECT * FROM Student WHERE<条件表达式>;
- **C1：无条件;**
- **C2：Sno='20180003';**
- **C3：Sbirthdate>='2000-1-1';**
- **C4：Smajor='计算机科学与技术' AND**
- **Sbirthdate>='2000-1-1'**

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t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10.1-C2] 假设Sno上有索引，索引扫描算法：
- 使用索引得到Sno为 '20180003'元组的指针
- 通过元组指针在Student表中检索到该学生

[例10.1-C3] 假设Sbirthdate上有B+树索引，索引扫描算法：
- 使用B+索引找到Sbirthdate>='2000-1-1'的第一个索引项，以此为入口点在B+树的顺序集上得到满足该条件的所有元组指针
- 再通过这些元组指针在Student表中检索到所有2000年1月1日以后出生的学生

[例10.1-C4] 假设Sbirthdate和Smajor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

4 4.2.2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10.2]** 针对如下SQL语句：

```
SELECT * FROM  Student, SC  WHERE Student.Sno = SC.Sno;
```

- 常用算法：
  - **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**
  - **哈希连接算法 (hash join)**
    - 划分阶段（创建阶段）
    - 试探阶段（连接阶段）

各算法具体步骤请参见教材

5 RDBMS的优化

- 查询优化在**RDBMS**中有着非常重要的地位
- 关系查询优化是影响**RDBMS**性能的主要因素
- 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

- 主要内容：
  - **查询优化概述**
  - **实例分析**

5.1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，因为：
    - 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

- 查询优化的计算：
  - 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

影响数据库系统性能的主要因素
新技术：内存数据库

- 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，尽量降低查询代价

5.2实例分析

- 以下通过一个简单的实例分析说明查询优化的必要性

[例10.3] 求选修了**81003**号课程的学生姓名

```
SELECT Student.Sname
FROM Student, SC
WHERE Student.Sno=SC.Sno AND SC.Cno='81003';
```

– 假设数据库中有**1000**个学生记录，**10000**个选课记录，其中选修**81003**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：

$$Q_1 = \Pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='81003'}(Student \times SC))$$

$$Q_2 = \Pi_{Sname}(\sigma_{SC.Cno='81003'}(Student \bowtie SC))$$

$$Q_3 = \Pi_{Sname}(Student \bowtie \sigma_{SC.Cno='81003'}(SC))$$

- 情形一:

$$Q_1 = \Pi_{Sname}(\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='81003'}(Student \times SC))$$

3.最后做投影 ← 2.再做选择 ← 1.先做笛卡尔积

- 计算步骤:

1. 计算 result1=Student×SC
2. 计算 result2=$\sigma_{Student.Sno=SC.Sno \wedge SC.Cno='81003'}$(result1)
3. 计算 result3=Π_{Sname}(result2)，得到最终结果

5 ~(cont.)

- 1.计算result1=Student×SC，算法如下：
  - 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 $\frac{1000}{10} + \frac{1000}{10\times5} \times \frac{10000}{100}$ =100+20×100=2100块
    - 连接后的元组数为$10^3\times10^4=10^7$。设每块能装10个元组，则写出10^6 块。
- 2.计算result2=$\sigma_{\text{Student.Sno=SC.Sno}\wedge\text{SC.Cno='81003'}}$(result1)，算法如下：
  - 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10^6块；
  - 若满足条件的元组假设仅50个，均可放在内存。
- 3.计算result3= Π_{Sname}(result2)
- 执行情形一查询的总读写数据块=$2100+10^6+10^6$

- 情形二:

$$Q_2 = \Pi_{Sname}(\sigma_{SC.Cno='81003'}(Student \bowtie SC))$$

- 算法:
  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，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

- 执行情形二总读写数据块=$2100+10^3+10^3$，执行代价约为情形一的488分之一

- 情形三:

$$Q_3 = \Pi_{Sname}(Student \bowtie \sigma_{SC.Cno='81003'}(SC))$$

- 算法:
  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

- 执行情形三总读写数据块=100+100，执行代价约为情形一的万分之一、情形二的二十分之一

- 进一步分析：
  - 假如SC表的Cno字段上有索引，第一步就不必读取所有SC元组而只需读取Cno= '81003'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，参加连接的元组就可以大大减少，这就是代数优化
  - 在Q_3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方法的初步概念

6 代数优化

厦門大學 XIAMEN UNIVERSITY

- SQL语句经过查询分析、查询检查后变换为语法树
  - 语法树是关系代数表达式的内部表示
- 基于关系代数等价变换规则的优化方法即为代数优化(或逻辑优化)
- 本节主要内容：
  - **关系代数表达式等价变换规则**
  - **语法树的启发式优化**

6.1关系代数表达式等价变换规则

- 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。
- 两个关系表达式**E1和E2是等价的**，记为**E1≡E2**
- **代数优化策略是通过对关系代数表达式的等价变换来提高查询效率**
- **11个**常用的等价变换规则见教材，请自行研读

6.2语法树的启发式优化

- **典型的启发式规则有：**
  - 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
  - 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
  - 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
  - 把某些选择同在它前面要执行的笛卡儿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儿积省很多时间
  - 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。
    - 当查询的对象是视图时，定义视图的表达式就是一种公共子表达式

- 语法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$\sigma_{F1\wedge F2\wedge\ldots\wedge Fn}(E)$的表达式变换为$\sigma_{F1}(\sigma_{F2}(\ldots(\sigma_{Fn}(E))\ldots))$。 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.把经过上述变换得到的语法树的内节点分组。每一双目运算（×，⋈，∪，－）和它所有的直接祖先为一组（这些直接祖先是单目运算σ或∏）。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（×），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10.4] 给出[例10.3]中 SQL语句的代数优化

```
SELECT Student.Sname
FROM Student, SC
WHERE Student.Sno=SC.Sno AND SC.Cno='81003';
```

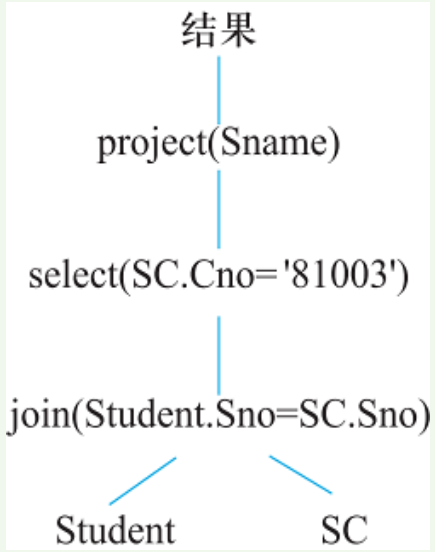


图10.3 语法树图

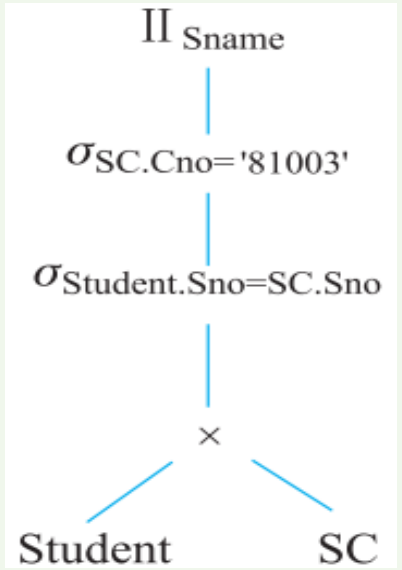


图10.4 关系代数语法树

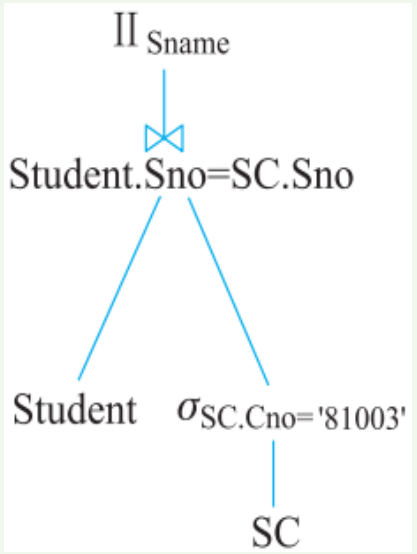
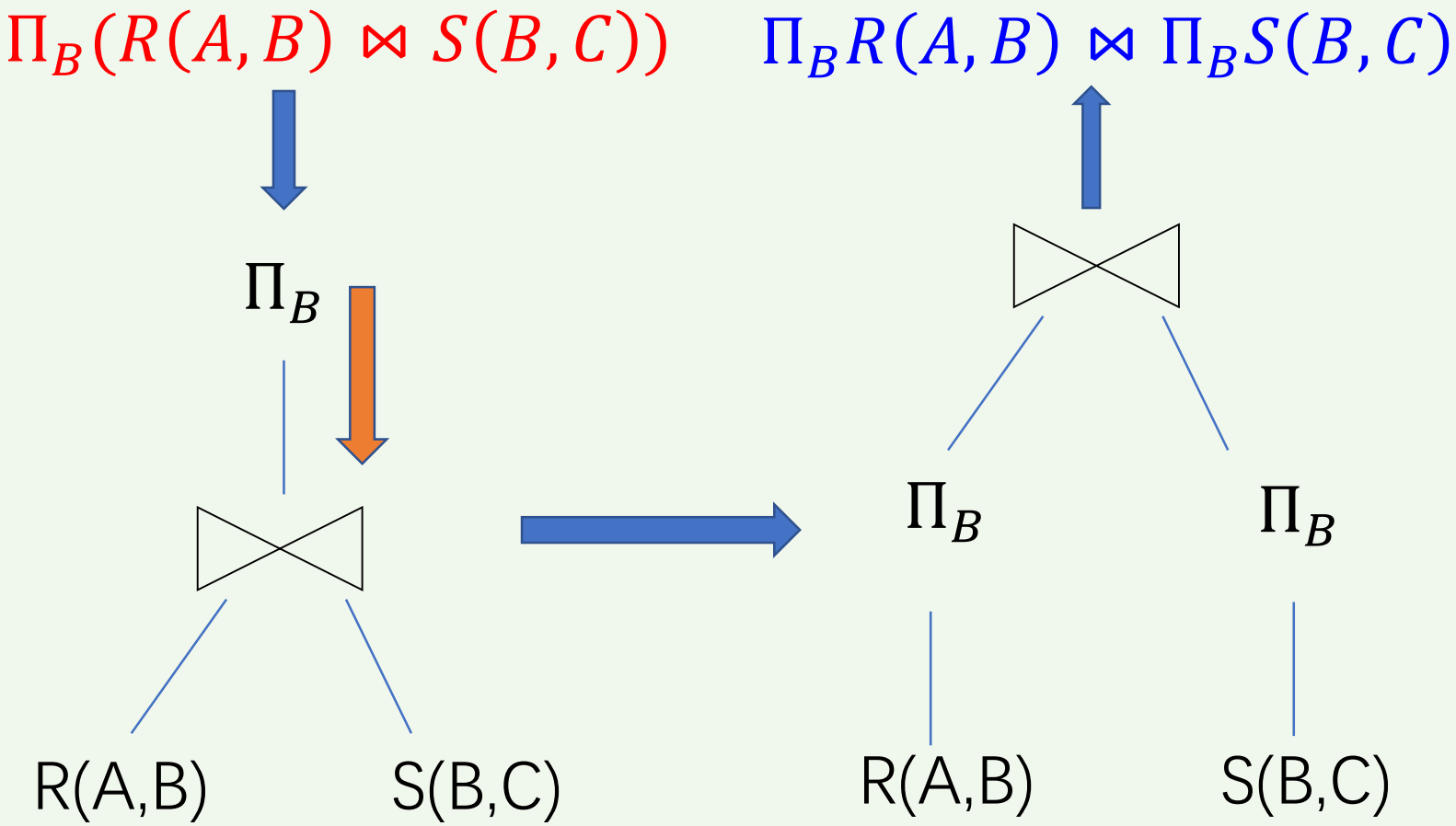


图10.5 优化后的语法树

- 问题：语法树图各节点与SQL语句各部分的对应关系？优化过程如何体现在图的变化中？

[例1] 对关系代数表达式$\Pi_B(R(A,B) \bowtie S(B,C))$进行查询优化



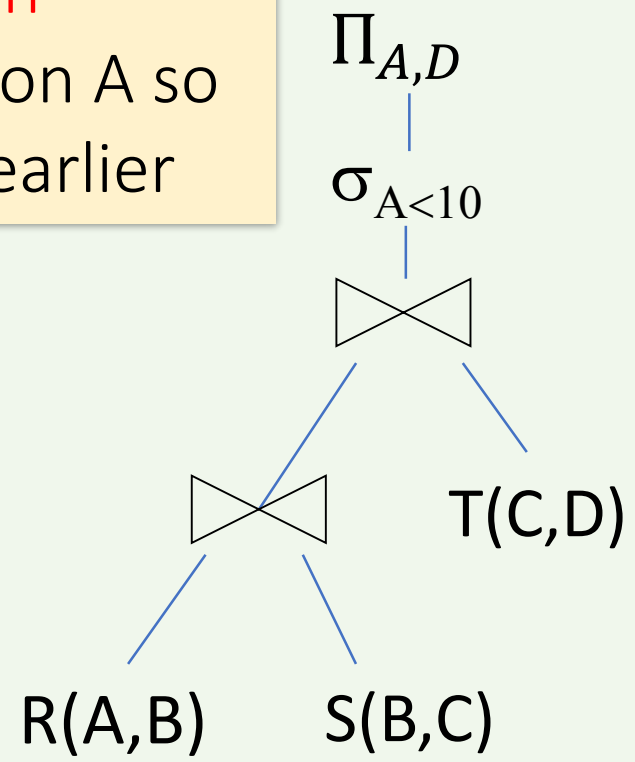
[例2]

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;
```

Push down selection on A so it occurs earlier

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



示例3(cont.)

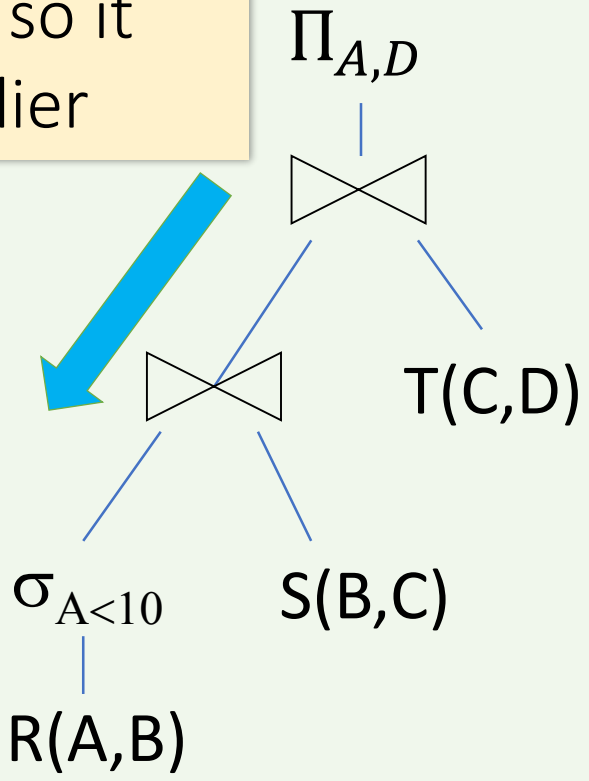
[例2]

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;
```

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

Push down projection so it occurs earlier



示例3(cont.)

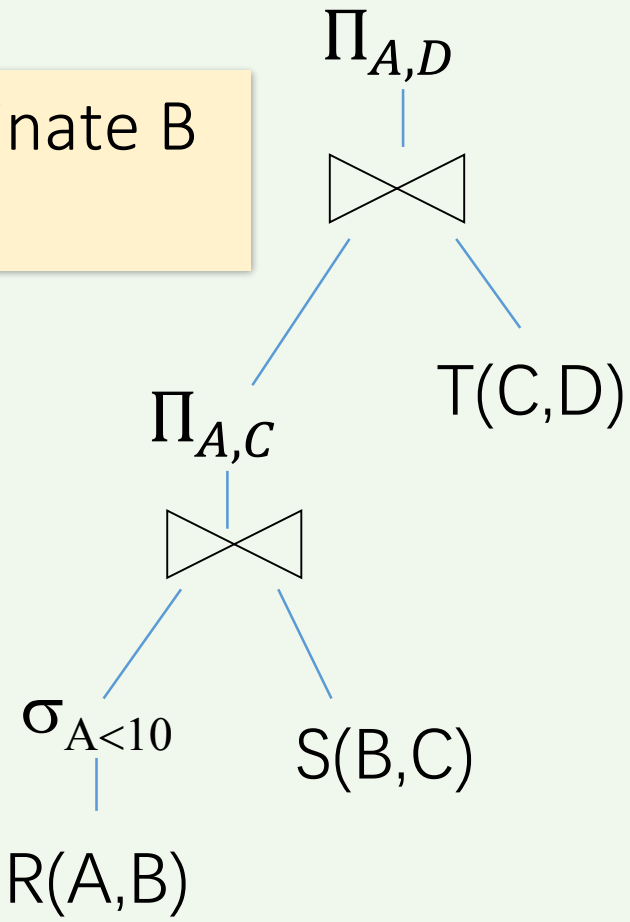
[例2]

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!



6 课堂练习

- 现有SQL语句如下：

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

请分别画出该SQL语句对应的初始语法树图、关系代数语法树和优化后的语法树

7 物理优化

厦門大學 XIAMEN UNIVERSITY

- 代数优化改变查询语句中操作的次序和组合，不涉及底层存取路径
- 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，求得优化查询计划
- **物理优化方法**
  - 基于启发式规则的启发式优化
    - 启发式规则是指那些在大多数情况下都适用但不是在每种情况下都是适用的规则
  - 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
  - 两者结合的优化方法
    - 常常先使用启发式规则，选取若干较优的候选方案，减少代价估算的工作量
    - 然后分别计算这些候选方案的执行代价，较快地选出最终的优化方案

- 主要内容：
  - **基于启发式规则的优化**
    - 选择操作的启发式规则
    - 连接操作的启发式规则
  - **基于代价的优化**

7.1基于启发式规则的优化

关系大小	选择操作的启发式规则
小关系	• 全表顺序扫描，即便选择列上有索引
大关系	1.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RDBMS会自动建立主码索引
	2.选择条件是“非主属性 = 值”或非等值或范围查询，且选择列上有索引的查询 • 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.使用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
	4.使用OR连接的析取选择条件 • 一般使用全表顺序扫描

条件	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.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)	• 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) • 显然应该选块数小的表作为外表

7.2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
  - 实现简单，优化本身的代价较小
  - 解释执行的系统，优化开销包含在查询总开销之中
  - 一次编译，多次执行
  - 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

1.统计信息

- 基于代价的优化方法要计算查询的各种不同执行方案的执行代价，它与数据库的状态密切相关，为此在数据字典中存储了优化器需要的数据库统计信息

优化器需要的统计信息		
对基本表的参数	**对基表列的参数**	**对索引的参数**
• 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)	• 该列不同值的个数(m) • 列最大值 • 最小值 • 列上是否已经建立了索引 • 哪种索引(B^+树,Hash,聚簇) • 可以计算选择率(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，则要计算每个值的选择率，f = 具有该值的元组数/N	• 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)

2.代价估算示例

– 全表扫描算法和索引扫描算法的代价估算

算法类型	代价估算
全表扫描算法	• 如果基本表大小为B块，代价为 cost =B。满足条件的元组占用f*B块
	• 如果选择条件是“码=值”，则平均搜索代价 cost=B/2 • 至多一条元组满足条件
索引扫描算法	• 如果选择条件是“码=值”，则采用该表的主码索引，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 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，索引的选择基数为N/m(有N/m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) cost=L+N/m
	• 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，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

2.代价估算示例

– 嵌套循环连接算法和排序-合并连接算法的代价估算

算法类型	代价估算
嵌套循环连接算法	• 代价 $cost = B_r + B_r B_s/(K-1)$
	• 如果需要把连接结果写回磁盘，则 $cost = B_r + B_r B_s/(K-1) + (F_{rs}*N_r*N_s)/M_{rs}$ • F_{rs}为连接选择性，表示连接结果元组数的比例， • M_{rs}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
排序合并连接算法	• 如果连接表已经按照连接属性排好序，则 $cost = B_r + B_s + (F_{rs}*N_r*N_s)/M_{rs}$
	• 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，其外排序的代价大约是4B，包括按可用内存大小分段读入文件(代价为B)，排成有序子表并写出去(代价为B)，最后将所有的有序子表归并成全局有序文件并写回磁盘(代价为2B) 。因此总代价为$cost=5B_r+5B_s+ (F_{rs}*N_r*N_s)/M_{rs}$盘 • 在内存足够大的情形下，也可以不生成全局有序文件，而是直接在有序子表上进行合并，这样可以节省2B的代价。

基于语义的查询优化

- 根据数据库的语义约束，把原来的查询转换成另一个执行效率更高的查询。

  考虑以下简单选择查询：

```
SELECT * FROM Student WHERE  Smajor='计算机科学与技术' AND  Sbirthdate>='2030-1-1';
```

  - 显然，用户误将出生日期值**Sbirthdate**由**’2000-1-1’**写成**’2030-1-1’**。假设数据库模式上定义了一个约束，要求学生年龄为**15-55**岁。一旦查询优化器检查到了这条约束，它就知道上面查询的结果为空，所以根本不用执行这个查询

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	执行过程
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完整个关系。 • 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

8 PostgreSQL示例

- 参考：PostgreSQL修炼之道之PostgreSQL中执行计划
  - https://zhengxianghang.blog.csdn.net/article/details/127225078

9 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

10 本章作业

- 教材第十章全部习题.
- 要求：作业布置后的一周内完成并提交到课程网站