清华大学数据库技术与应用

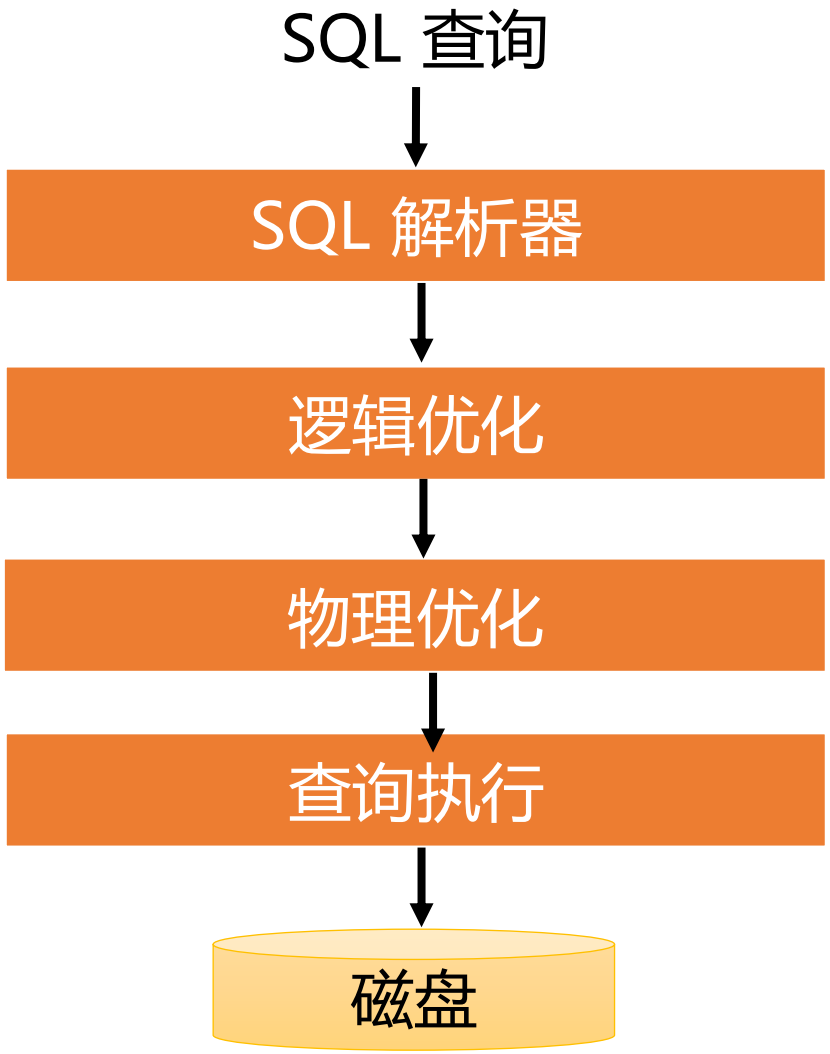
数据库性能优化 II

授课教师：计算机系王健楠

授课学期：2026年（春季）

清華大學 Tsinghua University

查询处理步骤



示例

- **Offering** (oID, dept, cNum, term, instructor)
- **Took** (sID, oID, grade)

问题：查询所有修读过 CMPT 354 课程的学生学号

```
SELECT sID
FROM   Offering O, Took T
WHERE  O.oID = T.oID
          AND O.dept =   'CMPT'
          AND O.cNum =   '354'
```

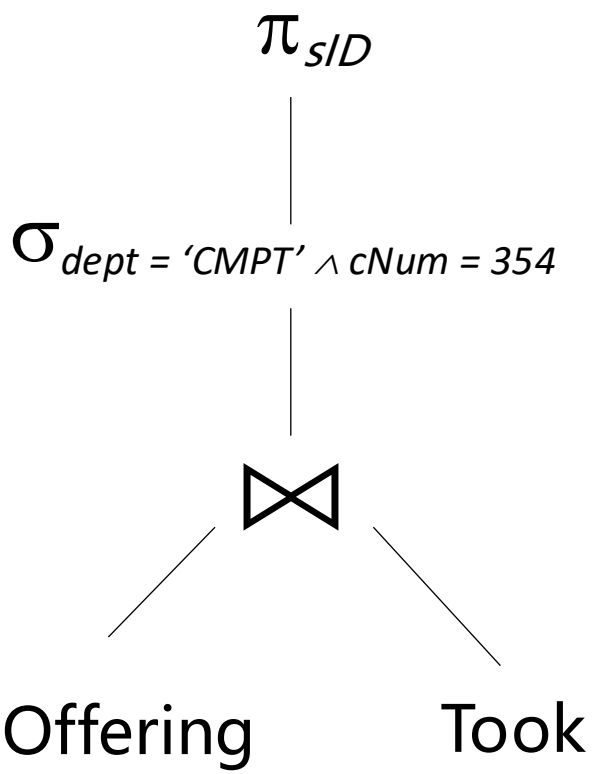
SQL 解析器

Offering (<u>oID</u>, dept, cNum, term, instructor)
Took (<u>sID, oID</u>, grade)

SQL

```
SELECT sID
FROM   Offering O, Took T
WHERE  O.oID = T.oID
       AND O.dept =   'CMPT'
       AND O.cNum =   '354'
```

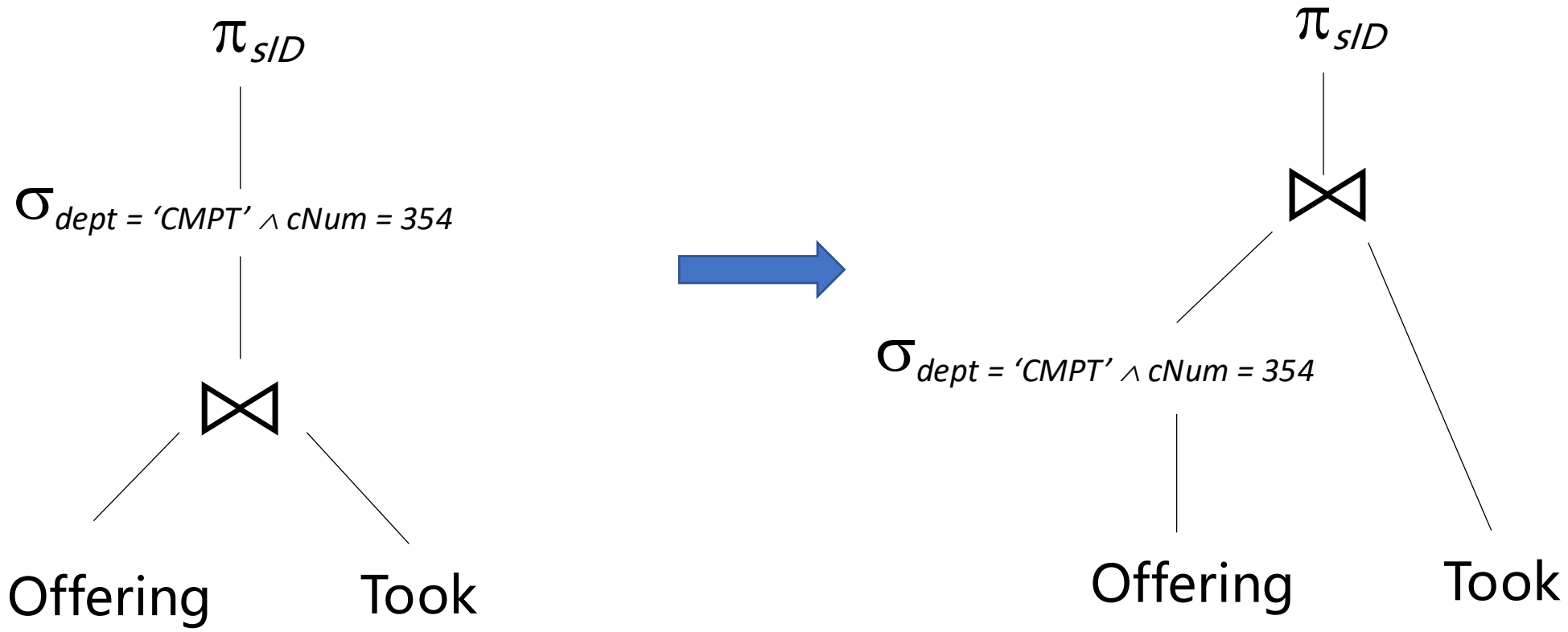
逻辑计划



逻辑优化

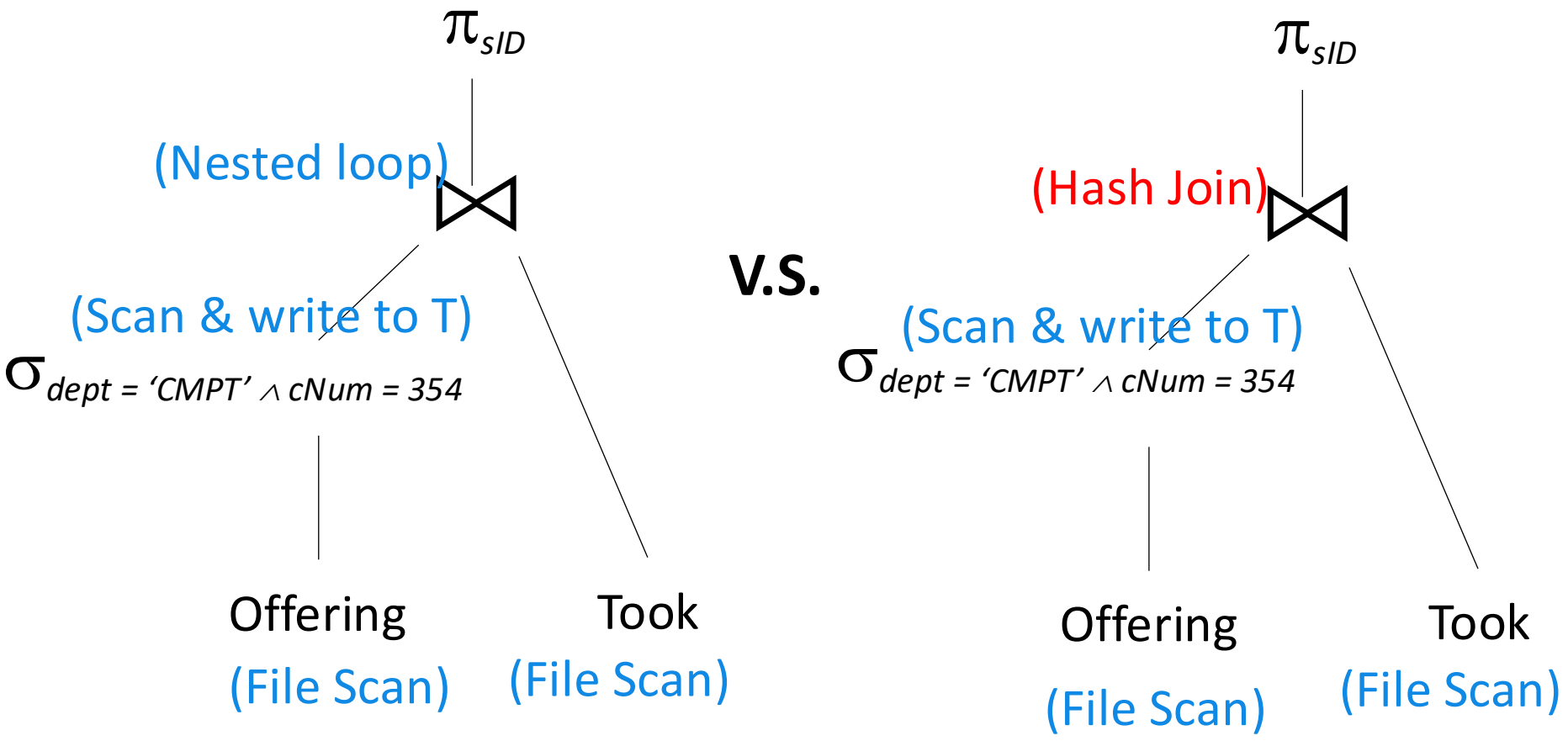
Offering (<u>oID</u>, dept, cNum, term, instructor)
Took (<u>sID, oID</u>, grade)

- 找到最优的**逻辑计划**



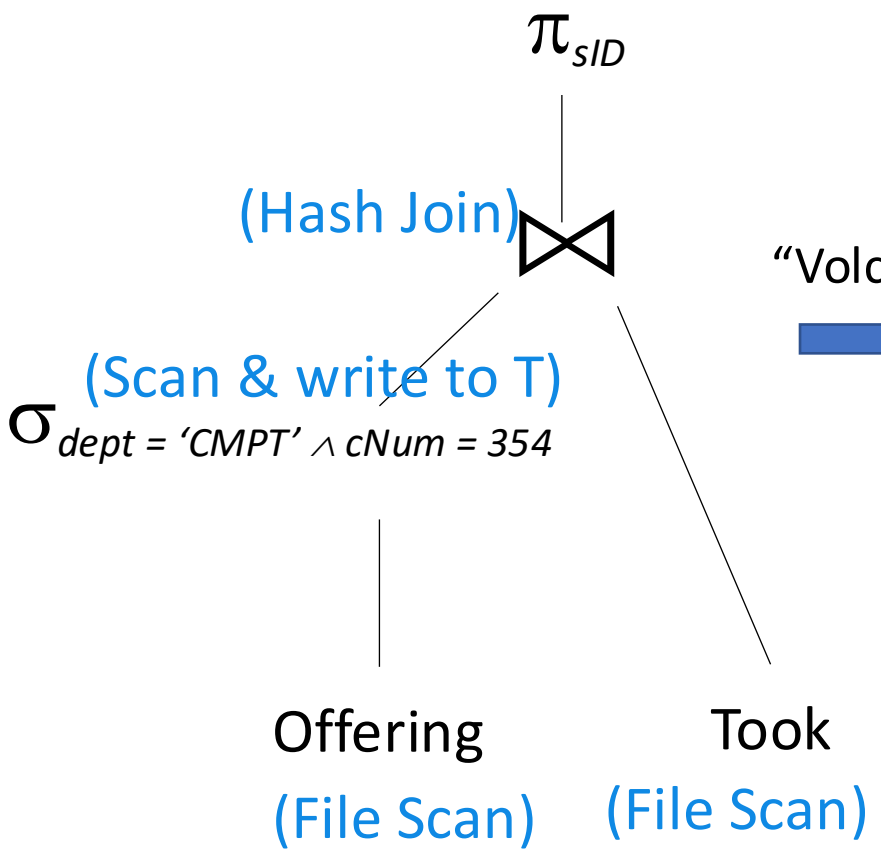
物理优化

- 找到最优的**物理计划**

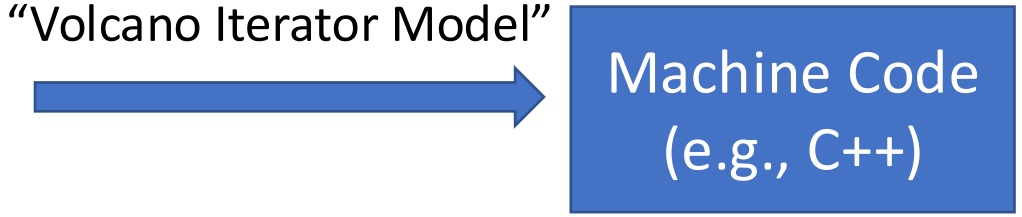


查询执行

物理计划



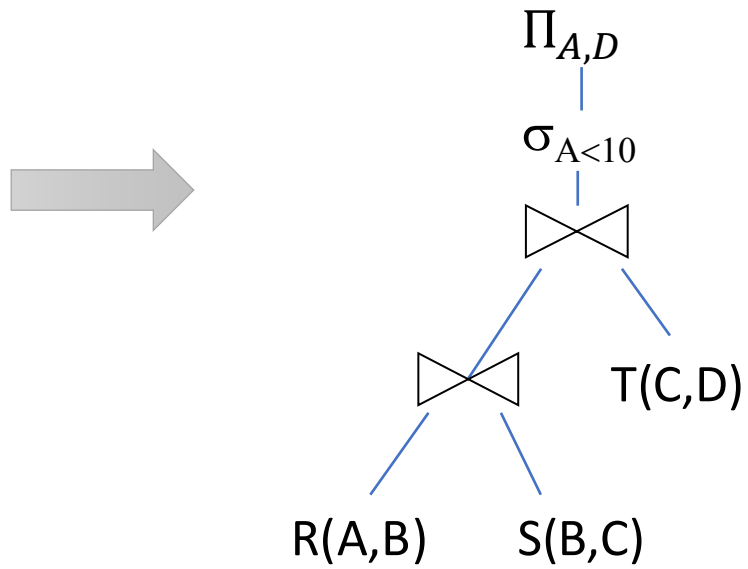
机器代码



练习

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

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

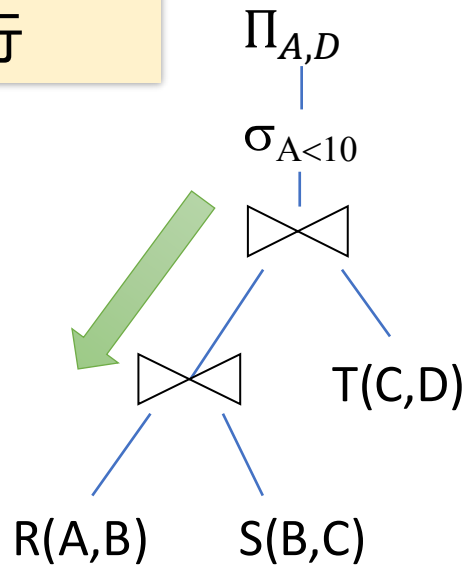


优化关系代数（RA）计划

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

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

将针对 A 的选择操作下推，使其更早执行

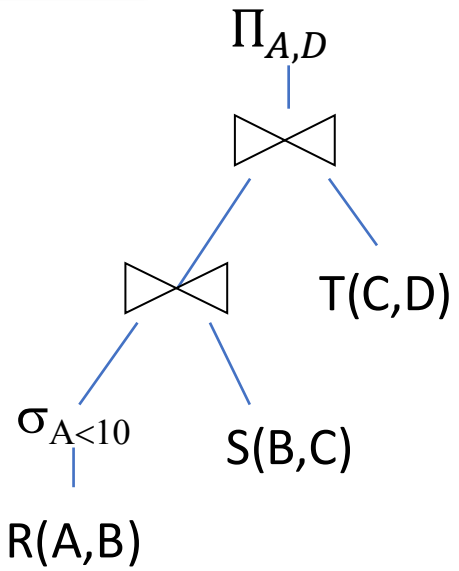


优化关系代数（RA）计划

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

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

将针对 A 的选择操作下推，使其更早执行

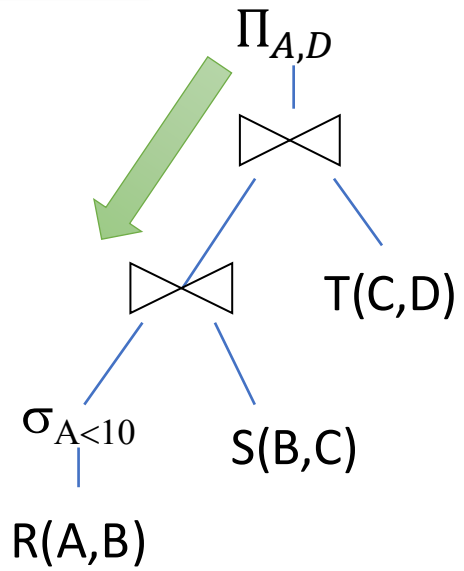


优化关系代数（RA）计划

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

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

将投影操作下推，使其更早执行

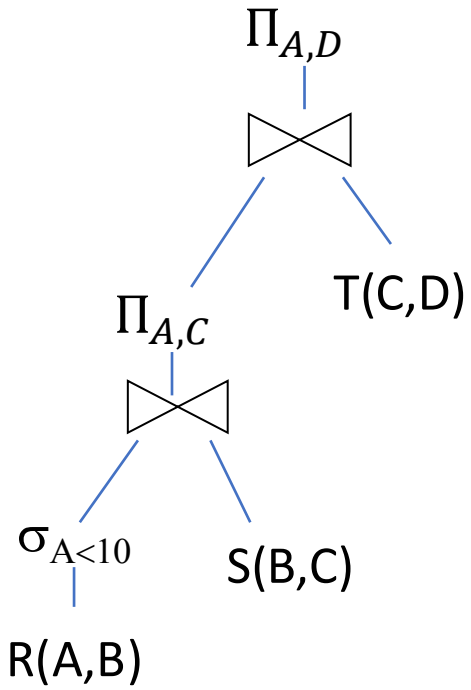


优化关系代数（RA）计划

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

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

可以更早消除属性 B!



IBM System R 优化器

清華大學 Tsinghua University

- 查询优化器的首次实现
- 让人们相信 DBMS 的性能可以超越人工优化
- 许多核心概念沿用至今

Access path selection in a relational database management system
https://dl.acm.org/citation.cfm?id=582099
by PG Selinger - 1979 - Cited by 2585 - Related articles
In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to **access paths**. This paper ...
Abstract · Authors · References · Cited By

如何构建查询优化器?

1. 查询计划空间
   – 枚举所有可能的查询计划

空间过大，必须剪枝

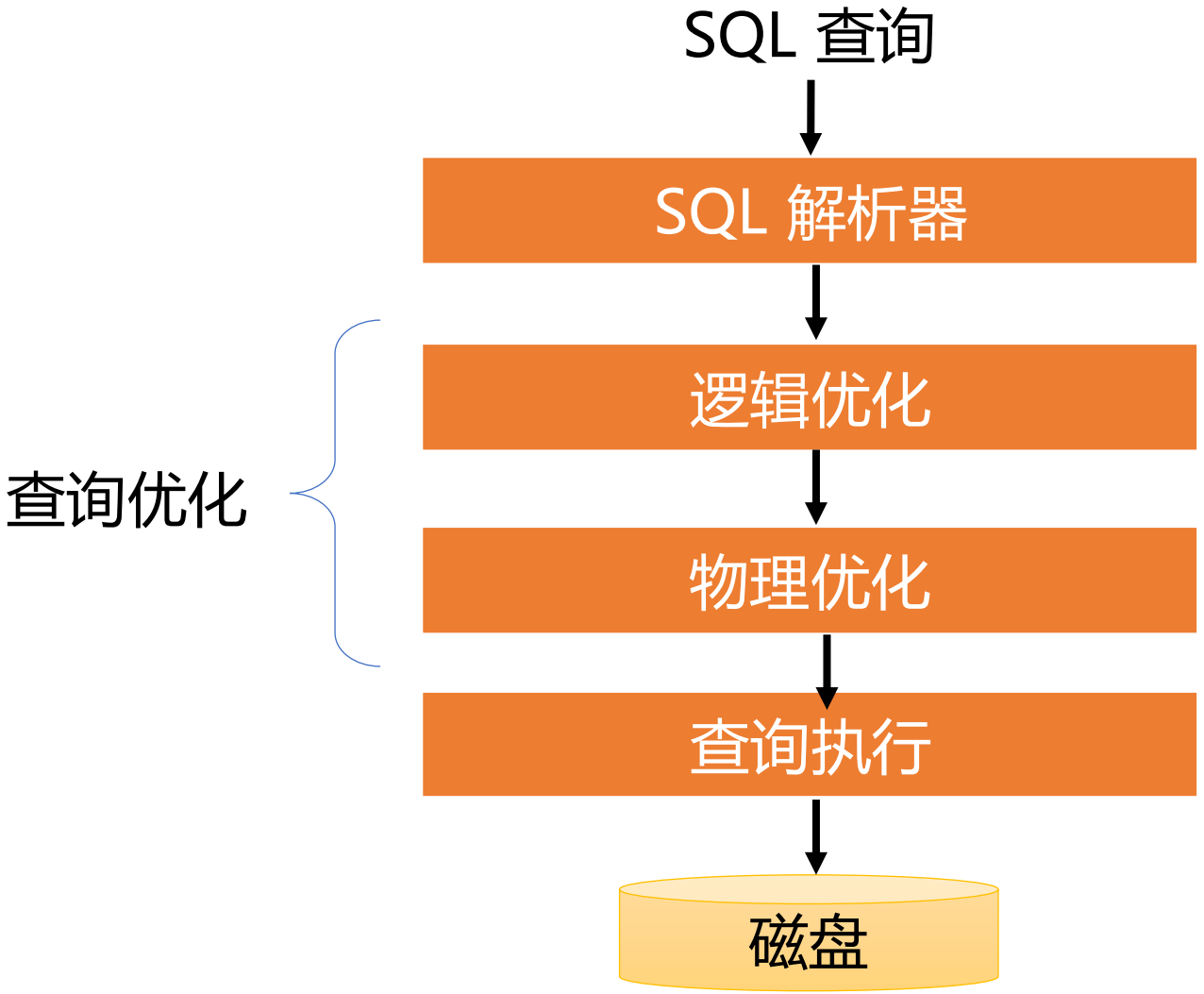
2. 代价估算
   – 估算每个计划的执行代价

CPU + I/O

3. 搜索算法
   – 寻找最优计划

不追求全局最优，寻找代价最小的可行计划

查询处理步骤



逻辑计划：

- 由解析器从输入的 SQL 文本生成
- 以关系代数树的形式表达
- 每个SQL有多重可能的逻辑计划

物理计划：

- 目标是为关系代数中每个算子选择高效的实现方式
- 每个逻辑计划对应多种可能的物理计划

查询优化：

- 找到最优逻辑查询计划
- 找到最优的物理查询计划

提纲

- **逻辑优化**
  - 选择下推
  - 投影下推
- **物理优化**
  - 连接算法
  - 选择率估算

提纲

- 逻辑优化回顾
  - 选择下推
  - 投影下推

- **物理优化**
  - 连接算法
  - 直方图

连接算法

- 嵌套循环连接
- 哈希连接
- 排序合并连接

Student

sname	uID
Mike	0
Joe	1
Alice	0
Marry	1
Bob	0
Tim	2

University

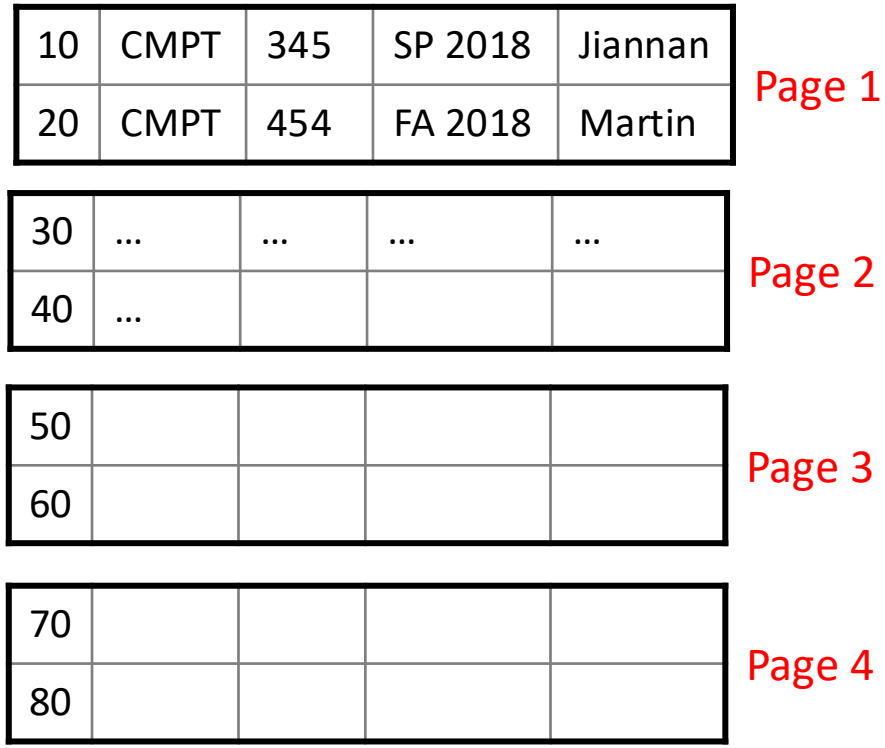
uID	uname
0	SFU
1	UBC
2	UT

Student ⋈ University

深入了解嵌套循环连接（NLJ）

基础知识

- 代价 = I/O + CPU + 网络
- "I/O 感知"算法
  - **本课程重点关注 I/O 代价**
- 对于关系 R，定义：
  - T(R) = R 中的元组数
  - P(R) = R 占用的页数



嵌套循环连接（NLJ）

Compute R ⋈ S on A:

```
for r in R:
  for s in S:
    if r[A] == s[A]:
      yield (r,s)
```

嵌套循环连接（NLJ）

Compute R ⋈ S on A:

```
for r in R:
  for s in S:
    if r[A] == s[A]:
      yield (r,s)
```

代价：

P(R)

1. 遍历 R 中的每个元组

注意：I/O 代价基于加载的页数，而非元组数！

嵌套循环连接（NLJ）

Compute R ⋈ S on A:

```
for r in R:
  for s in S:
    if r[A] == s[A]:
      yield (r,s)
```

代价：

P(R) + T(R)*P(S)

1. 遍历 R 中的每个元组

2. **对 R 中的每个元组，遍历 S 中的所有元组**

R 中每个元组都需要从磁盘读取一次完整的 S！

嵌套循环连接（NLJ）

Compute R ⋈ S on A:

```
for r in R:
  for s in S:
    if r[A] == s[A]:
      yield (r,s)
```

代价：

P(R) + T(R)*P(S)

1. 遍历 R 中的每个元组

2. 对 R 中的每个元组，遍历 S 中的所有元组

3. 检查是否满足连接条件

注意：NLJ 不仅限于等值连接，可在 if 语句中检查任意条件！

嵌套循环连接（NLJ）

Compute R ⋈ S on A:

```
for r in R:
  for s in S:
    if r[A] == s[A]:
      yield (r,s)
```

这与笛卡尔积有何不同?

代价：

P(R) + T(R)*P(S) + OUT

1. 遍历 R 中的每个元组
2. 对 R 中的每个元组，遍历 S 中的所有元组
3. 检查是否满足连接条件
4. **写出结果（先写入缓冲页，页满后写入磁盘）**

嵌套循环连接（NLJ）

Compute R ⋈ S on A:

```
for r in R:
  for s in S:
    if r[A] == s[A]:
      yield (r,s)
```

代价：

P(R) + T(R)*P(S) + OUT

若将 R（外表）与 S（内表）互换，结果如何?

P(*S*) + T(*S*)*P(*R*) + OUT

外表与内表的选择对性能影响巨大——DBMS 需要知道哪个关系更小!

I/O 感知方法

块嵌套循环连接（BNLJ）

Compute R ⋈ S on A:

```
for each B-1 pages pr of R:
  for page ps of S:
    for each tuple r in pr:
      for each tuple s in ps:
        if r[A] == s[A]:
          yield (r,s)
```

假设内存中有 B+1 个缓冲页

代价：

$\mathrm{P}(R)$

1. **每次将 R 的 B-1 页读入内存（各留 1 页给 S 和输出缓冲）**

注意：顺序读取多页可能带来额外加速，此处忽略不计！

块嵌套循环连接（BNLJ）

Compute R ⋈ S on A:

```
for each B-1 pages pr of R:
  for page ps of S:
    for each tuple r in pr:
      for each tuple s in ps:
        if r[A] == s[A]:
          yield (r,s)
```

假设内存中有 B+1 个缓冲页

代价：

$$\mathrm{P}(R) + \frac{P(R)}{B-1} P(S)$$

1. 每次将 R 的 B-1 页读入内存（各留 1 页给 S 和输出缓冲）

2. **对 R 的每个 (B-1) 页块，逐页读入 S 进行比较**

注意：优先遍历较小的关系速度更快！

块嵌套循环连接（BNLJ）

Compute R ⋈ S on A:

```
for each B-1 pages pr of R:
  for page ps of S:
    for each tuple r in pr:
      for each tuple s in ps:
        if r[A] == s[A]:
          yield (r,s)
```

假设内存中有 B+1 个缓冲页

代价：

$$\mathrm{P}(R) + \frac{P(R)}{B-1} P(S)$$

1. 每次将 R 的 B-1 页读入内存（各留 1 页给 S 和输出缓冲）
2. 对 R 的每个 (B-1) 页块，逐页读入 S 进行比较
3. **Check against the join conditions**

BNLJ 同样支持非等值连接条件

块嵌套循环连接（BNLJ）

Compute R ⋈ S on A:

```
for each B-1 pages pr of R:
  for page ps of S:
    for each tuple r in pr:
      for each tuple s in ps:
        if r[A] == s[A]:
          yield (r,s)
```

假设内存中有 B+1 个缓冲页

代价：

$$\mathrm{P}(R) + \frac{P(R)}{B-1} P(S) + \mathrm{OUT}$$

1. 每次将 R 的 B-1 页读入内存（各留 1 页给 S 和输出缓冲）
2. 对 R 的每个 (B-1) 页块，逐页读入 S 进行比较
3. Check against the join conditions
4. **写出结果**

BNLJ 与 NLJ 对比：I/O 感知的优势

- NLJ
  - 每处理 R 的一个元组，就需要从磁盘读取一次完整的 S
- BNLJ
  - 每处理 R 的一个 (B-1) 页块，才需要从磁盘读取一次完整的 S

NLJ

P(R) + T(R)*P(S) + OUT

BNLJ

$$\mathrm{P}(R) + \frac{P(R)}{B-1} P(S) + \mathrm{OUT}$$

BNLJ 比 NLJ 大约快 $\frac{(B-1)T(R)}{P(R)}$ 倍!

BNLJ 与 NLJ 对比：I/O 感知的优势

- 示例数据：
  - R：500 页
  - S：1000 页
  - 每页 100 个元组
  - 内存大小为 12 页（B = 11）

此处忽略输出代价…

- NLJ 代价 = 500 + **50,000×1000 = 5000 万次 I/O**
- $\frac{500*1000}{10}$BNLJ 代价 = 500 + 50×1000 = **5 万次 I/O**

算法上的微小改动，带来实实在在的性能差异！

提纲

- 逻辑优化回顾
  - 选择下推
  - 投影下推
- 物理优化
  - 连接算法
  - 直方图

动机

- 假设在 name 字段建立索引，运行以下查询

```
SELECT sID
FROM Student
WHERE  name = ?
```

查询优化器会使用索引吗？
会！

- 假设在 gender 字段建立索引，运行以下查询

```
SELECT sID
FROM Student
WHERE  gender = ?
```

查询优化器会使用索引吗？
不会！

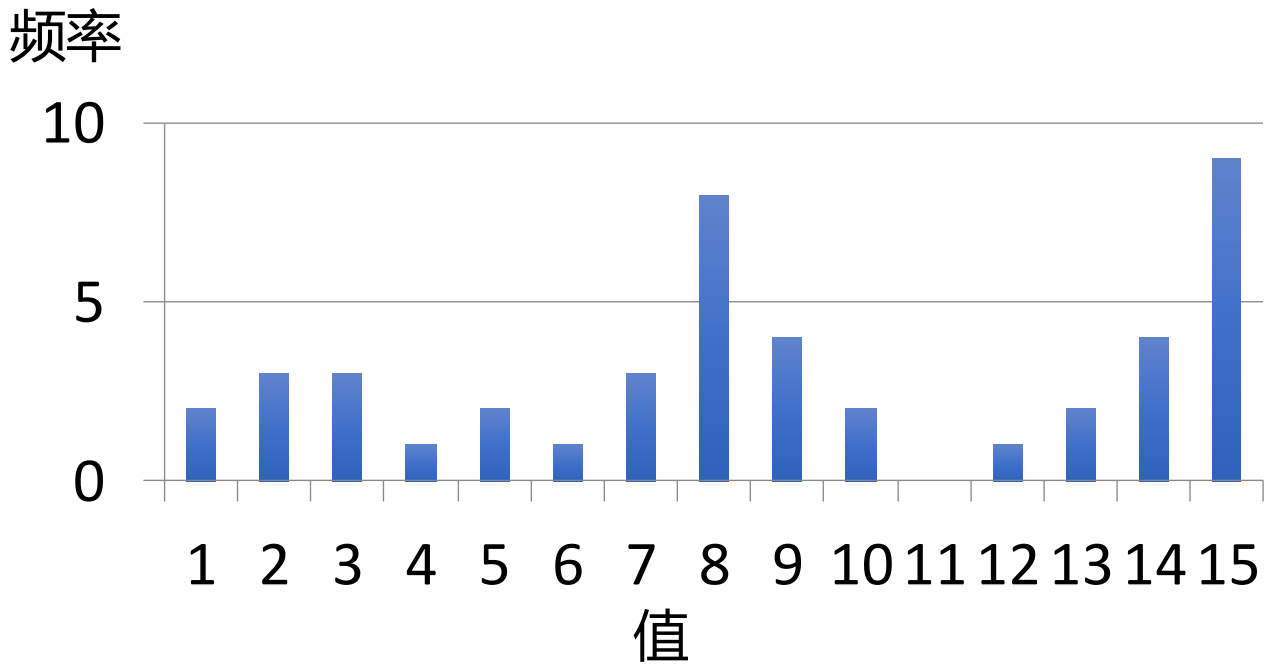
查询优化器如何判断是否使用索引？

直方图（Histograms）

清華大學 Tsinghua University

- 直方图是一组值域范围（"桶"）及各桶中值出现频率的统计
- 如何划分桶（bucket）？
  - 等宽（Equiwidth）与等深（Equidepth）
- 高频值对代价估算尤为重要

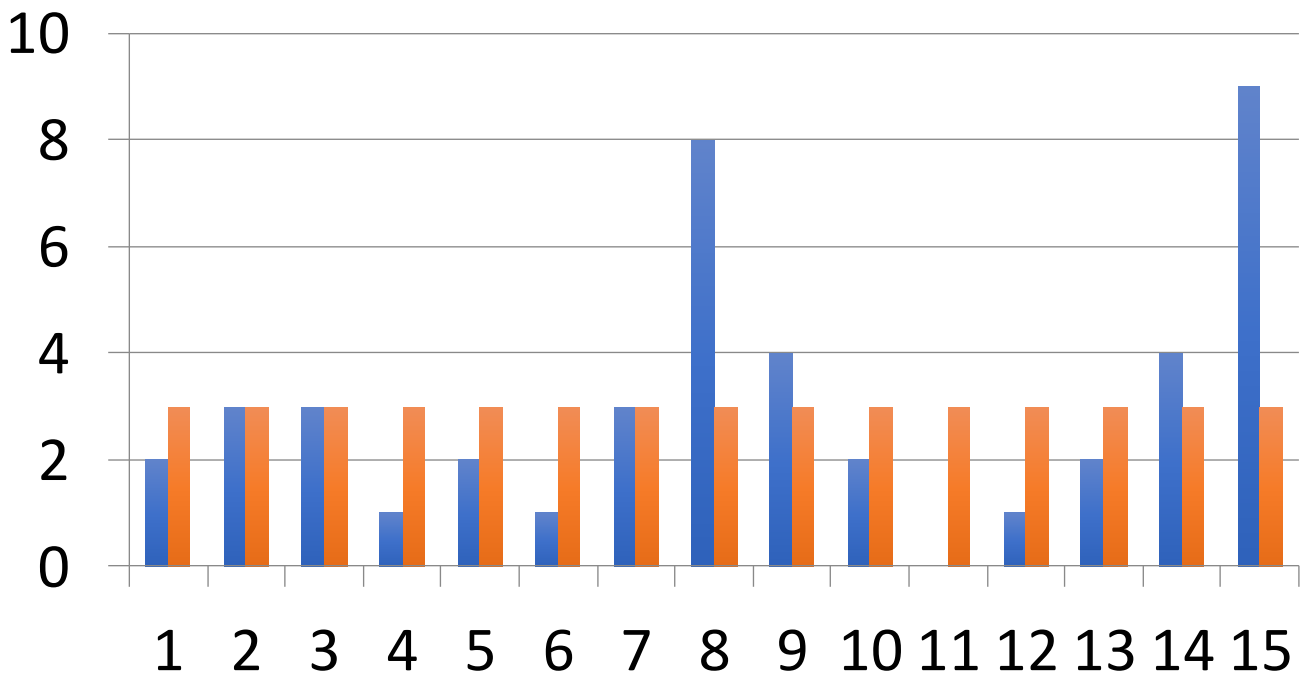
示例



如何计算 8 到 10 之间有多少个值?
(答案显而易见)

问题：精确计数占用空间太大!

完整计数 vs. 均匀计数



完整计数（桶大小=1）需要多少存储空间?

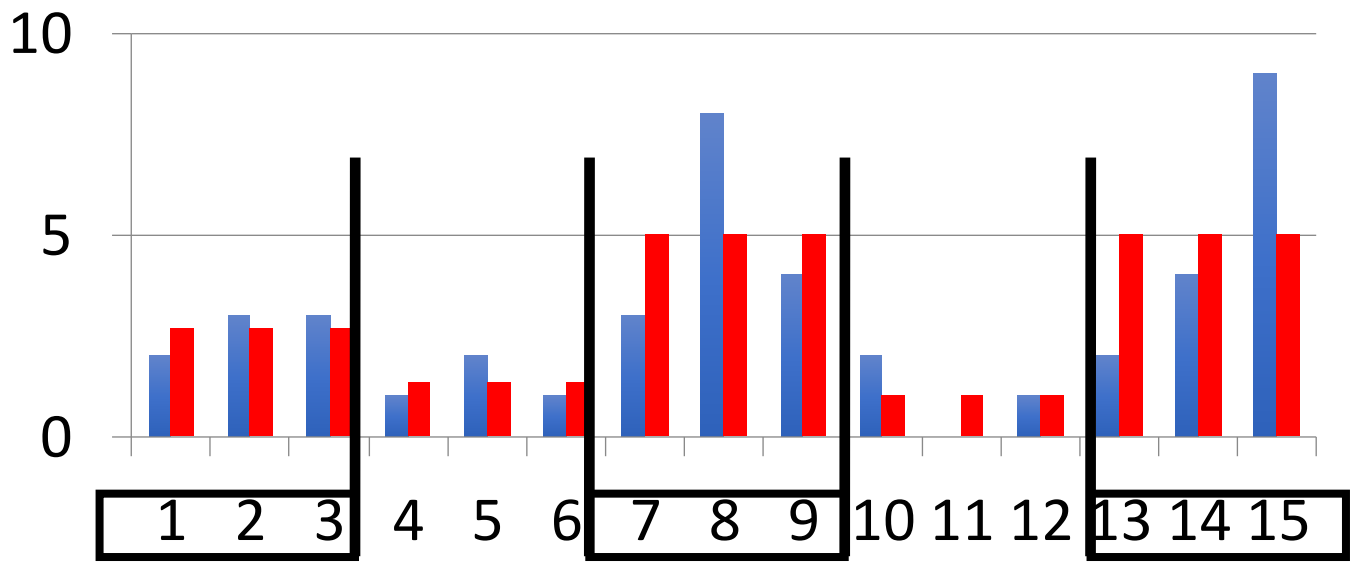
均匀计数（桶大小=全部）需要多少存储空间?

基本权衡

- 希望精度高（如完整计数）
- 希望存储空间小（如均匀计数）
- 直方图是两者的折中方案！

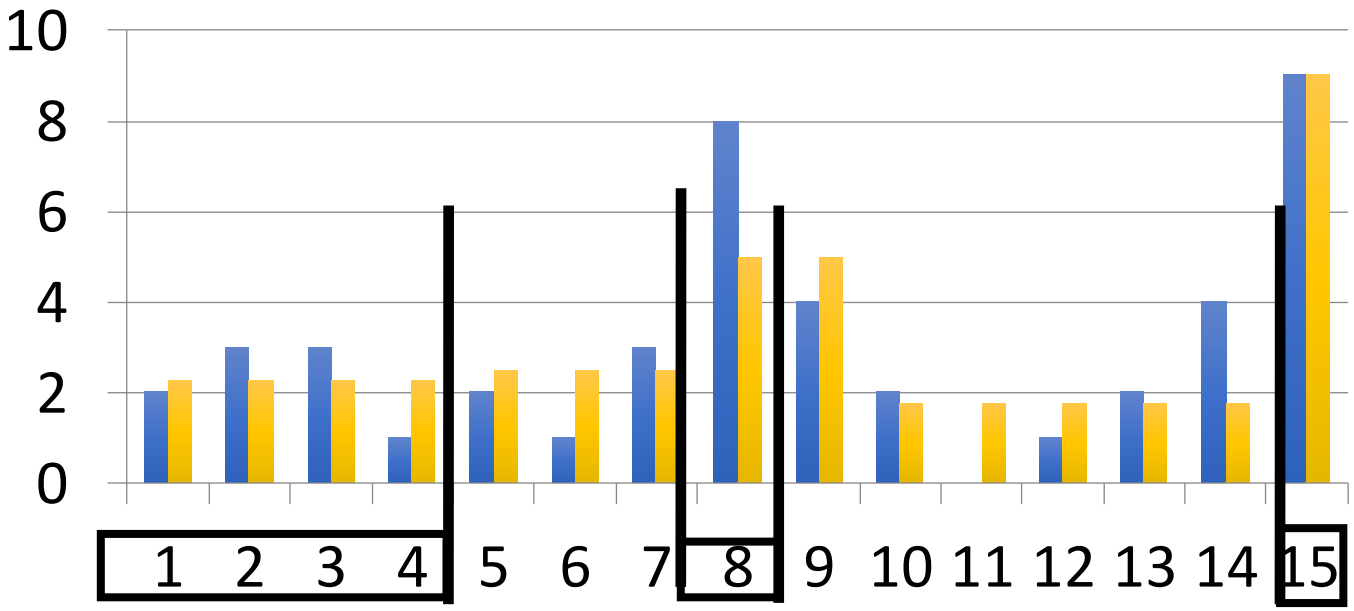
那么如何确定"桶"的大小？

等宽直方图



所有桶的宽度大致相同

等深直方图



所有桶包含大致相同数量的元素
(总频率相等)

直方图

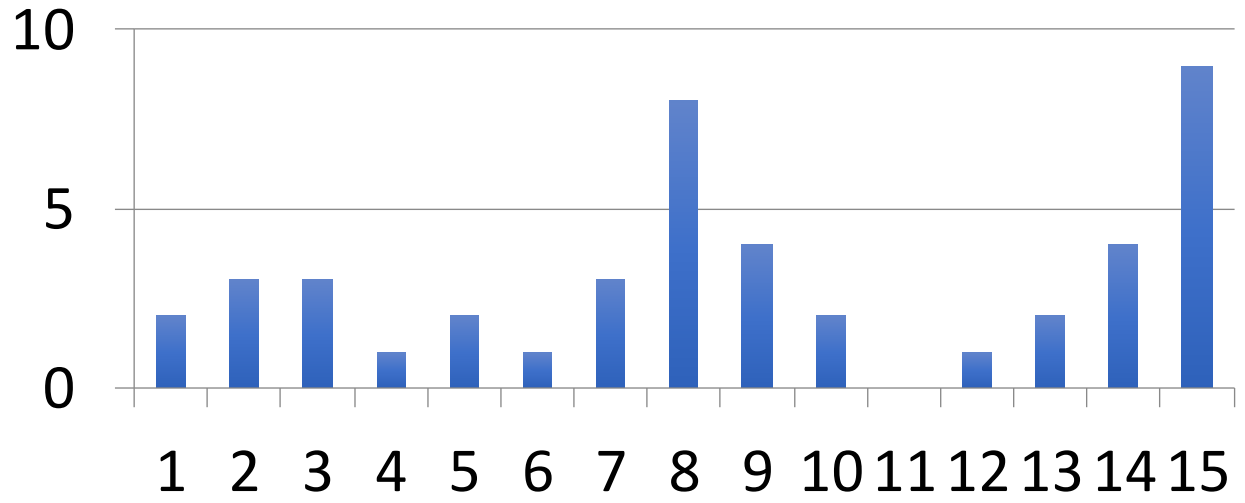
- 简单、直观且广泛使用
- 参数：桶的数量与类型
- 可扩展至多属性（多维直方图）

直方图的维护

清華大學 Tsinghua University

- 直方图需要及时更新!
  - 通常需要手动执行或定期调度命令来更新数据库统计信息
  - 过期的直方图可能导致严重的性能问题!
- 目前已有关于自适应直方图与查询反馈的研究工作
  - Oracle 11g（已支持此功能）

棘手的示例



1. 插入大量 value > 16 的元组
2. 不更新直方图
3. 查询 value > 20 的结果?

压缩直方图

一种常见方法：

1. 显式存储最高频的值及其计数
2. 对其余值保留等宽或等深直方图

研究者还尝试了各种高级方法：小波变换、图模型、熵模型等...

总结

- **逻辑优化**
  - SQL → RA Tree
  - 选择下推
  - 投影下推
- **物理优化**
  - 嵌套循环连接（NLJ）/ 哈希连接 / 排序合并连接
  - I/O 感知算法
  - 直方图（Histogram）