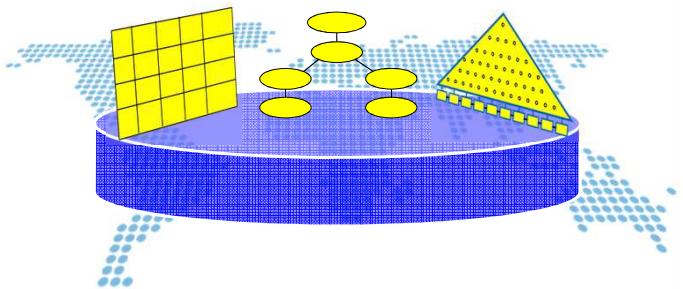
数据库系统

查询优化

陈世敏

（中科院计算所）

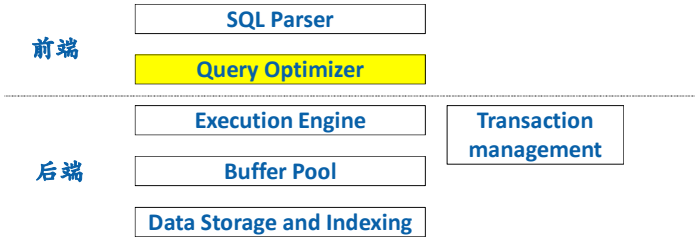


Outline

- 查询优化概述
- 将SQL查询转换成关系代数表达式
- 关系代数的等价变换
- 运算代价的估计
- 基于成本的计划选择
- 多个查询块：嵌套子查询、视图、集合运算

查询优化概述

- SQL内部表达➔Query Plan (执行方案)
  - 产生可行的query plan
  - 估计query plan的运行时间和空间代价
  - 在多个可行的query plans中选择最佳的query plan



语法分析

- 对SQL语句进行语法分析
- 形成语法树
  - class, struct等组成的树
  - 例如，包含select list, from list, where conditions, group list, having conditions, order by list等
- 并完成预处理
  - 检查SQL语句中的关系：必须为table或view
  - 检查SQL语句中的属性列：必须出现在引用的table或view中，而且没有歧义（没有同名的问题）
  - 检查数据类型：运算或比较操作、操作数的类型一致
- 把view视图的定义放入
  - 后面会举例介绍

查询优化的步骤

- 产生初始执行方案
  - 将查询分解为块
    - 在语法树的基础上
    - 每块包含一个SELECT-FROM，至多有一个WHERE、GROUP BY和HAVING语句
  - 每块查询➔关系代数表达式
  - 子查询和视图：结合多块查询
- 产生等价方案：关系代数表达式等价变换
- 代价估计：对于每种执行方案进行代价估计
- 寻找最优方案：搜索执行方案空间

举例：假设下面的关系模式

- create table Sailors(sid integer primary key,
  sname varchar(20) unique, rating integer, age real);
- create table Boats(bid integer primary key,
  bname varchar(20), color varchar(10));
- create table Reserves(sid integer, bid integer, day date
  primary key (sid, bid, day),
  foreign key (sid) references Sailors(sid),
  foreign key (bid) references Boats(bid)
  );

举例：对预订了至少两艘红色船并且评价等于所有水手中最高评价的水手，输出其标识和预订红色船只的最早时间

```
select S.sid, MIN(R.day)
from Sailors S, Reserves R, Boats B
where S.sid=R.sid and R.bid=B.bid and B.color='red'
  and S.rating = (select MAX(S2.rating)
                  from Sailors S2)
group by S.sid
Having COUNT(*)>=2;
```


将SQL查询分解为块

- 每块包含一个SELECT-FROM，至多有一个WHERE、GROUP BY和HAVING语句

```
select S.sid, MIN(R.day)
from Sailors S, Reserves R, Boats B
where S.sid=R.sid and R.bid=B.bid and B.color='red'
  and S.rating = 嵌套内层查询的引用
group by S.sid
Having COUNT(*)>=2;
```

```
select MAX(S2.rating)
from Sailors S2
```

Outline

- 查询优化概述
- 将SQL查询转换成关系代数表达式
- 关系代数的等价变换
- 运算代价的估计
- 基于成本的计划选择
- 多个查询块：嵌套子查询、视图、集合运算

转化一个查询块（无嵌套）

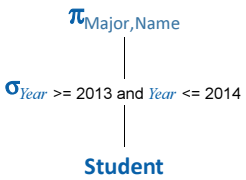
select	投影π，去重δ，聚集γ
from	选择σ，或者连接⋈ （先用X表示）
where	选择σ，或者连接⋈
group by	分组γ
having	选择σ
order by	排序τ

通常使用一个事先定义好的基本方法产生关系代数表达式

举例

显示2013-2014年入学的学生的系和姓名

```
select Major, Name
from Student
where Year >= 2013 and Year <= 2014;
```

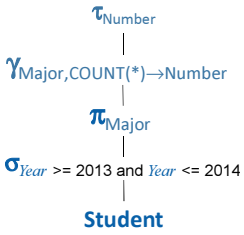


举例

统计各系2013-2014年入学的学生人数，并按照人数排序

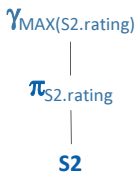
```
select Major, count(*) as Number
from Student
where Year >= 2013 and Year <= 2014
group by Major
order by Number;
```

注意：统一地在选择投影之后使用group-by等，投影的列为所有输出列和中间使用的列



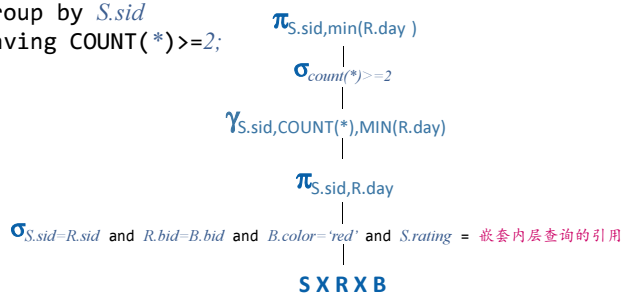
举例

```
select MAX(S2.rating)
from Sailors S2
```



举例

```
select S.sid, MIN(R.day)
from Sailors S, Reserves R, Boats B
where S.sid=R.sid and R.bid=B.bid and B.color='red'
  and S.rating = 嵌套内层查询的引用
group by S.sid
Having COUNT(*)>=2;
```



转化一个查询块（无嵌套）的步骤

- 通用的方法：单个查询块转化为关系代数形式
- 把from, where语句直接转化为σ π X 的形式
  - from成为一个关系或多个关系的X（先不写成连接形式）
  - where是σ的条件
  - π的投影列是SELECT语句中其它子句中出现的列
- 在此基础上，添加group by, having, order by等

Outline

- 查询优化概述
- 将SQL查询转换成关系代数表达式
- 关系代数的等价变换
- 运算代价的估计
- 基于成本的计划选择
- 多个查询块：嵌套子查询、视图、集合运算

关系代数等价变换的目的

- 通过等价变换，可以得到等价的执行方案
- 每种执行方案的代价不同
- 目的是得到代价较小的关系代数形式
  - 例如：希望尽早完成过滤操作以减小中间结果的大小

AND选择条件的分解

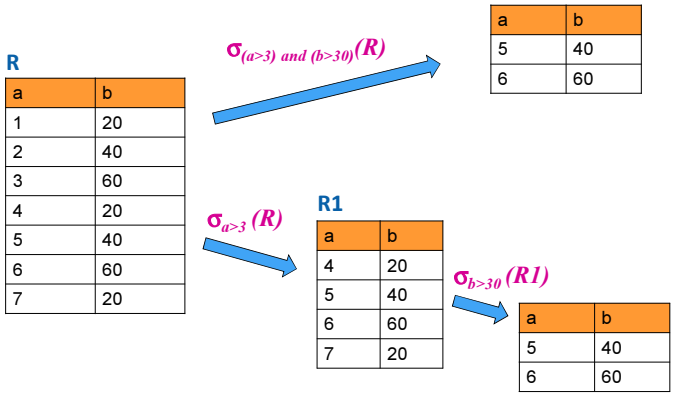
- 以AND连起来的选择条件可以分别计算

$$\sigma_{C1\ and\ C2}(R) = \sigma_{C1}(\sigma_{C2}(R))$$

- 选择条件的求解顺序是灵活的
  （对应于 and满足交换律）

$$\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$$

AND选择条件的分解



OR选择条件可以分解吗？

$\sigma_{C1\ or\ C2}(R) = ?$

$\sigma_{C1\ or\ C2}(R) = \sigma_{C1}(R) \cup \sigma_{C2}(R)?$

- 当C1与C2没有公共记录时成立
- 当C1与C2有公共记录时，R必须为集合，∪必须为集合运算进行去重才成立
- 否则，右侧可能比左侧多一些重复的记录

OR选择条件的分解

R

a	b
1	20
2	40
3	60
4	20
5	40
6	60
7	20

$\sigma_{(a>3)\ or\ (b>30)}(R)$

a	b
2	40
3	60
4	20
5	40
6	60
7	20

$\sigma_{a>3}(R)$

a	b
4	20
5	40
6	60
7	20

$\sigma_{b>30}(R)$

a	b
2	40
3	60
5	40
6	60

U

a	b
4	20
5	40
6	60
7	20
2	40
3	60
5	40
6	60

错误！多了一些记录

$\sigma_{(a>3)\ or\ (b>30)}(R)$!= $\sigma_{a>3}(R) \cup \sigma_{b>30}(R)$

OR选择条件的分解

R

a	b
1	20
2	40
3	60
4	20
5	40
6	60
7	20

$\sigma_{(a<2)\ or\ (b>30)}(R)$

a	b
1	20
2	40
3	60
5	40
6	60

$\sigma_{a<2}(R)$

a	b
1	20

$\sigma_{b>30}(R)$

a	b
2	40
3	60
5	40
6	60

U

a	b
1	20
2	40
3	60
5	40
6	60

分解正确！没有交集

$\sigma_{(a<2)\ or\ (b>30)}(R)$ == $\sigma_{a<2}(R) \cup \sigma_{b>30}(R)$

分解选择条件

- 例：设R(a,b,c)是一个关系，那么$\sigma_{(a=1\ or\ a=3)and\ b<c}(R)$如何分解？
- $\sigma_{(a=1\ or\ a=3)and\ b<c}(R) = \sigma_{(a=1\ or\ a=3)}(\sigma_{b<c}(R))$
  $= \sigma_{a=1}(\sigma_{b<c}(R)) \cup \sigma_{a=3}(\sigma_{b<c}(R))$

或者

- $\sigma_{(a=1\ or\ a=3)}(\sigma_{b<c}(R)) = \sigma_{b<c}(\sigma_{(a=1\ or\ a=3)}(R))$
  $= \sigma_{b<c}(\sigma_{a=1}(R) \cup \sigma_{a=3}(R))$

选择+集合运算的等价变换

- 集合运算（并交差）要求schema完全一致
  - 具有相同的列数
  - 相同位置的列的名字和类型都一样
- 所以，选择条件可以应用到每个关系上
- 那么
  - $\sigma_C(R \cup S) = ?$
  - $\sigma_C(R \cap S) = ?$
  - $\sigma_C(R - S) = ?$

选择对于集合运算

- $\sigma_C(R \cup S)$
  - $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
- $\sigma_C(R \cap S)$
  - $\sigma_C(R \cap S) = \sigma_C(R) \cap \sigma_C(S)$
  - $\sigma_C(R \cap S) = \sigma_C(R) \cap S$
  - $\sigma_C(R \cap S) = R \cap \sigma_C(S)$
- $\sigma_C(R - S)$
  - $\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S)$
  - $\sigma_C(R - S) = \sigma_C(R) - S$

举例算一下

- $\sigma_C(R \cup S)$
  - $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$
- $\sigma_C(R \cap S)$
  - $\sigma_C(R \cap S) = \sigma_C(R) \cap \sigma_C(S)$
  - $\sigma_C(R \cap S) = \sigma_C(R) \cap S$
  - $\sigma_C(R \cap S) = R \cap \sigma_C(S)$
- $\sigma_C(R - S)$
  - $\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S)$
  - $\sigma_C(R - S) = \sigma_C(R) - S$

- 例如：R和S的schema包含一个列col
- R.col={1,2,3,4,5,6}, S.col= {2,4,6,8,10}
- 条件C为 col > 4
- 我们算一下这些运算规律是否正确

选择+叉积/连接的等价变形

- σ_C(多个关系表的叉积或连接)
- 讨论选择条件的多种情况
  - 选择条件仅涉及其中一个表
  - 选择条件涉及多个表，也就是某种连接条件
    - 等值连接
    - 其它条件

举例

$\sigma_{S.sid=R.sid \text{ and } R.bid=B.bid \text{ and } B.color='red' \text{ and } S.rating = \text{嵌套内层查询的引用}}$

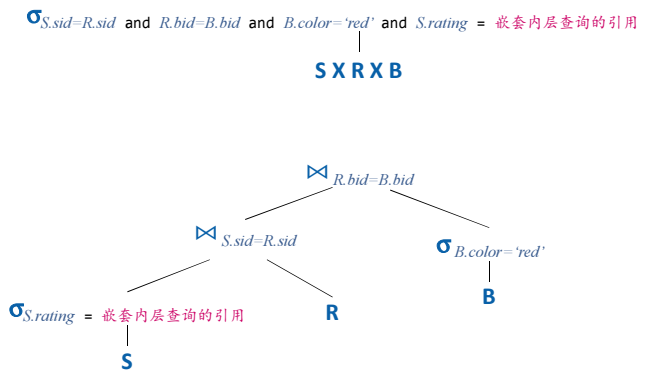
S X R X B

- S与R进行等值连接，连接条件是*S.sid=R.sid*
- R与B进行等值连接，连接条件是*R.bid=B.bid*
- 在B上进行选择：*B.color='red'*
- 在S上进行选择：*S.rating* = 嵌套内层查询的引用

优化1：下推选择

- 分解选择条件，尽可能把选择下推
  - 因为越早选择，越快地丢弃不相关的记录
  - 减少后续处理的代价
- 单表选择可以下推

举例



投影的等价变换

可以在下层引入新的投影操作

$\pi_{a,b}(R) = \pi_{a,b}(\pi_{a,b,c}(R))$

实际上

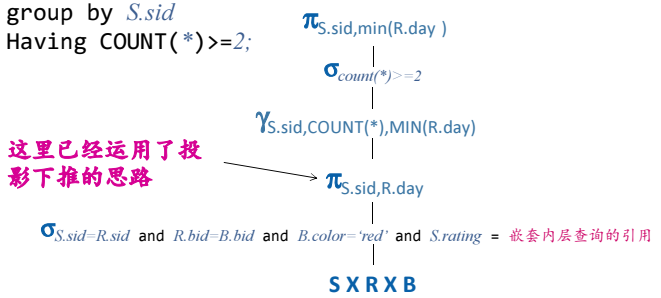
- 投影列包含上层用到的所有的列
- 那么仍然保证上层的操作不变

优化2：下推投影

- 把投影操作下推
  - 尽早丢弃不相关的列
  - 减少内存的占用，减少中间结果的大小
- 通常情况下
  - 一个表上面的第一个运算就是下推的选择
  - 再上面一个运算就是下推的投影

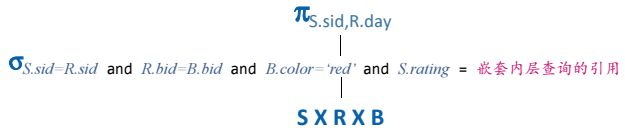
举例

```
select S.sid, MIN(R.day)
from Sailors S, Reserves R, Boats B
where S.sid=R.sid and R.bid=B.bid and B.color='red'
  and S.rating = 嵌套内层查询的引用
group by S.sid
Having COUNT(*)>=2;
```



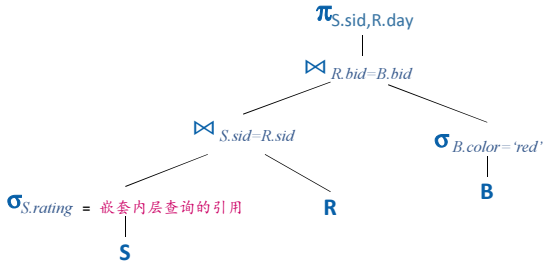
举例

关注最下面的选择、投影、连接运算



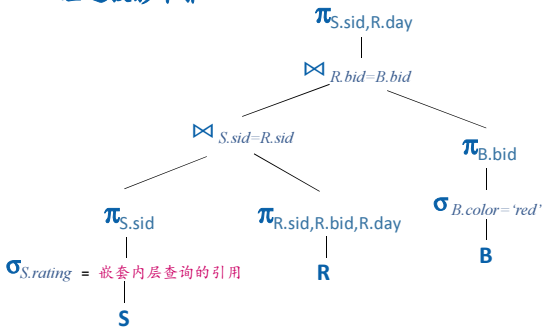
举例

经过选择下推



举例

经过投影下推



叉积和连接的等价变换

- 叉积和连接满足交换律和结合律
- 交换律
  - R⋈S = S⋈R
  - 改变两个关系表在连接算法中的作用
    - nested loop: outer relation, inner relation
    - hash join: build relation, probe relation
- 结合律
  - (R⋈S)⋈T= S⋈(R⋈T)
  - 可以改变连接的顺序，不同的顺序产生的中间结果大小通常不同，有不同的代价
- 等价变换：改变顺序

去重的等价变换

- 去重δ
  - 如果A是一个包，那么δ(A)就是一个集合
  - 相当于SQL的Distinct
- 什么样的关系表没有重复记录？
- 什么操作的结果肯定没有重复记录？

什么样的关系表没有重复记录？

- 定义了主键的关系表没有重复记录
  - 主键是unique而且not null
  - 多以每个记录的主键都不同
  - 没有重复记录
- 如果只有unique，没有主键？
  - 不行，可能为null
  - 可能有两个相同记录，这一列都为null

什么操作的结果肯定没有重复记录？

- (去重操作之外)
- group by操作肯定没有重复记录
  - 同一组的记录都放到了一起，进行了聚集
- SQL的union, intersect, except
  - 实现集合的并交差
  - 进行了去重操作

在上述两种情况下，可以省略δ

- 如果可以确定关系表R中没有重复记录，那么
  - $\delta(R) = R$
- 如果可以确定在某个操作后没有重复记录，那么
  - $\delta(\gamma...) = \gamma...$

去重可以下推的情况

- 下推去重的好处
  - 去重可以减少中间结果
- $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$
  - 可以先去重再选择
  - 选择不会产生新的重复记录
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
  - 可以先去重再连接
  - 连接不会产生新的重复记录
  - 也适用于叉积

去重不能下推的情况

- 投影
  - $\delta(\pi_C(R)) \neq \pi_C(\delta(R))$
  - 投影操作减少了列，可能使本来不同的记录变成重复记录
  - 所以，不可以把去重放到投影内
- 如前可知，投影被广泛采用，所以去重在很多情况下都不能下推
- 其它：包的并、差

分组聚集的等价变换

- 去重：在分组之后的去重，可以省略
  - $\delta(\gamma_L(R)) = \gamma_L(R)$
- 投影：分组之前可以引入投影，删除不相关的列
  - 只有分组的列，和聚集使用的列才需要保留
  - 参见前面有关投影的部分
- 一些聚集操作（例如MIN, MAX），可以先去重
  - 去重之后，不影响MIN, MAX的结果
  - 但是，这对于SUM, COUNT, AVG不成立

Outline

- 查询优化概述
- 将SQL查询转换成关系代数表达式
- 关系代数的等价变换
- 运算代价的估计
- 基于成本的计划选择
- 多个查询块：嵌套子查询、视图、集合运算

逻辑执行计划 vs. 物理执行计划

- 逻辑执行计划
  - 把SQL语句块改写为关系代数形式
  - 然后采用前述运算规律优化：下推选择、下推投影等
  - 产生的执行计划是逻辑执行计划
- 物理执行计划
  - 确定具体的数据访问路径
  - 确定具体的关系代数运算实现算法
  - 确定满足结合律分配律运算顺序：例如多个连接的顺序

逻辑执行计划➔物理执行计划

- 查询优化希望能够找到最优的物理执行计划

优化目标

min cost(物理执行计划)

限制条件

物理执行计划是给定逻辑执行计划的一个可行的实现

估计cost(物理执行计划)

- 运算代价估计主要涉及三个问题
  - 如何收集统计信息
  - 如何估计运算的代价
  - 如何估计结果的大小

如何收集统计信息?

- 代价估计中可能需要许多数据的统计信息
  - 关系表的大小：有多少数据页，多少记录
  - 记录的平均长度
  - 属性长度：定长，变长（平均长度）
  - 属性值的分布：有多少不同值、或者统计直方图
    - 如果在这个属性上有索引，可以记录索引中不同键的个数
    - 来确定属性值的个数

如何收集统计信息?

- 通常有下述方式收集统计信息
  - 静态收集
    - 数据库系统支持特殊的统计信息收集命令
      - DBA手工运行命令，收集统计信息
    - 数据库系统定期自动运行统计信息收集命令
    - 问题
      - 在收集时是比较精确的（也可能进行采样等方法近似）
      - 收集后随着增删改，会逐渐不准确
  - 动态收集统计信息
    - 有些信息可以动态收集，例如记录数
    - 希望统计信息的收集操作不影响系统正常运行时的性能
    - 方法：统计信息的收集在不加锁的情况下直接累计
      - 结果可能是不精确的

如何估计运算的代价?

- 查询处理中已经介绍
- 对于每种运算算法，建立运算代价模型
  - 选择
  - 连接算法：nested loop, hash join, sort merge join
  - 排序
  - 等
- 给定模型参数，就可以对运算代价进行估计

如何估计结果的大小?

- 为什么需要对结果进行估计?
  - 收集的统计信息是原始表的
  - 运算的代价估计需要输入数据的统计信息
  - 但是中间运算?
    - 运算的输入不一定是原始表
    - 可以是前一个运算的输出
- 下面重点讨论结果大小估计

结果大小估算的基本思路

- 使用准确的统计信息
  - 如果存在
- 进行简化假设
  - 包括独立同分布、经验值

投影：结果大小的估计

- 记录的数量：不变
- 记录的长度：变小
  - 根据属性长度
    - 定长、变长的平均长度
  - 额外空间：记录头等
- 可以估计完成投影后数据的大小

举例

- 假设R(a,b,c)是一个关系
  - a是长度为4B的整数
  - b是长度为4B的整数
  - c是平均长度为100B的字符串
  - 假设记录头采用前面课程的格式
    - 长度：2B
- 那么，原始记录平均长度？
  - 2+4+4+100=110B
- $\pi_{a,b}(R)$中记录的平均长度？
  - 2+4+4=10B

选择：结果大小的估计

- selectivity (选择度)
  - 满足选择条件的记录占总输入记录的比例
  - 换言之，也就是满足选择条件的概率
- 如果已知输入记录数Nrecord和选择度selectivity
  - 那么输出记录数可以估计为：Nrecord * selectivity
  - 假设Nrecord已知，那么主要需要估计selectivity
- 下面我们根据具体的选择条件进行讨论

Column=value

- 已知：属性的取值个数为Nkey
- 假设：属性值是随机均匀分布，Uniform distribution
  - 每个取值的记录数大致相同
- selectivity=1/Nkey

Column < value

- >, >=, <, <=不等值比较
- 方法一
  - 数值类型的列
  - 已知：maxkey, minkey
  - 假设：取值在[minkey, maxkey]上均匀分布
  - 那么估计selectivity为：$\frac{value - minkey}{maxkey - minkey}$
- 方法二
  - 考虑满足条件和不满足条件的记录
  - 通常情况下，一个选择条件过滤的记录多，留下的记录少
  - 满足条件的记录通常小于一半
  - 那么估计selectivity为：1/3

Column != value

- 不等值的比较
  - 认为不等的情况很少
  - 所以，输出的记录数与输入记录数相同
  - selectivity=1

Column1=Column2

- 已知：Column1上的取值个数为Nkey1
- 已知：Column2上的取值个数为Nkey2
- 假设：两列上的取值分布是独立的，每个列上取值的分布本身是均匀的
- 那么selectivity=$\frac{1}{\max(Nkey1, Nkey2)}$

Column1=Column2

- 如果只知道一个列的取值数为Nkey
- 那么selectivity估计为=$\frac{1}{Nkey}$
- 如果两个列的取值数都未知
- 那么selectivity估计为=$\frac{1}{10}$

Column in (…)

- 可以看作是column=value的扩展
  - 估计column=value的大小
  - 然后乘以in列表中的值的个数
  - 列表值的个数为Nlist, 列的取值数为Nkey
  - 估计selectivity为 $\frac{Nlist}{Nkey}$
- 额外的启发规则
  - 预期每个选择条件至少把输入记录数减少一半
  - 那么估计为min($\frac{Nlist}{Nkey}$, $\frac{1}{2}$)

条件1 AND 条件2

- 假设：两个条件是独立的
- 那么selectivity = selectivity1 * selectivity2

条件1 OR 条件2

- 假设：两个条件是独立的
- 那么可以估计同时不满足两个条件的概率为
  - (1-selectivity1) * (1-selectivity2)
- 那么至少满足其中一个条件的概率为
  - 1 - (1-selectivity1) * (1-selectivity2)

NOT 条件1

- 如果条件1的选择度为selectivity1
- 那么NOT 条件1的选择度为
  - 1 – selectivity1

等值连接：结果大小的估计

- $R \bowtie_{R.a=S.b} S$
- 先看一些特例

等值连接：结果大小的估计

- 情况1：主键和外键之间的连接
  - $R \bowtie_{R.a=S.b} S$
  - R.a是R外键，引用S.b，S.b是S的主键
  - 假设：每个输入的R记录都可以找到匹配
  - 输出记录数 = R的记录数
- 情况2：所有记录的R.a与S.b都相同
  - 那么就是叉积
  - 记录数=R的记录数 * S的记录数
- 通常情况如何估计？

等值连接：结果大小的估计

- $R \bowtie_{R.a=S.b} S$
  - 可以认为是在RxS叉积上的选择
  - 选择的条件是column1 = column2
  - 假设R的记录数为N_R， S的记录数为N_S
  - R.a的取值个数为$Nkey_R$，S.b的取值个数为$Nkey_S$
  - 那么输出匹配记录数估计为：$N_R * N_S / \max(Nkey_R, Nkey_S)$

举例

- 已知
  - R(a,b)记录数为1000条，b列取值有20个
  - S(b,c)记录数为2000条，b列取值有50个，c列取值有100个
  - T(c,d)记录数为5000条，c列取值有500个
- 估计 $R\bowtie_{R.b=S.b}S\bowtie_{S.c=T.c}T$的结果记录数？
  - 1000*2000*5000/(50*500) = 400,000

非等值连接

- 仍然可以看作是
  - 先叉积：输出的记录数为输入记录数的积
  - 再选择：采用选择条件selectivity的估计方法

并交差：结果大小估计

- 集合并 RUS
  - 设R的记录数为N_R，S的记录数为N_S
  - 最少输出记录数为$\max(N_R, N_S)$
  - 最多输出记录数为$N_R + N_S$
  - 可以取中值：$0.5*(\max(N_R, N_S) + N_R + N_S)$
- 集合交R∩S
  - 最少输出记录数为0
  - 最多输出记录数为$\min(N_R, N_S)$
  - 可以取中值：$0.5*\min(N_R, N_S)$
- 集合差R−S
  - 最少输出记录数为$N_R - N_S$
  - 最多输出记录数为N_R
  - 可以取中值：$0.5*(2N_R - N_S)$

去重：结果大小的估计

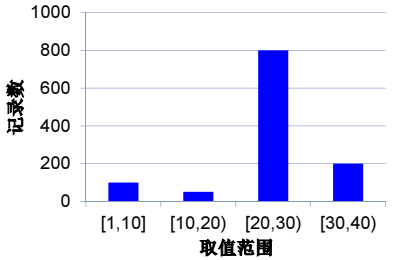
- 上限一：输入记录数
  - 假设R的记录数为N_R
  - 那么$\delta(N_R) <= N_R$
  - 估计时，认为大致有50%的记录被去重
- 上限二：各列取值数的积
  - 假设R有k个列，第i列的取值数为$Nkey_i$
  - 那么$\delta(N_R) <= Nkey_1*Nkey_2 * \ldots * Nkey_k$
- 估计为$0.5\ N_R$和$Nkey_1*Nkey_2 * \ldots * Nkey_k$的较小值

分组与聚集：结果大小的估计

- 上限一：输入记录数
  - 估计为输入记录数的一半，0.5Nrecord
- 上限二：分组的组数
  - 假设按照k个列分组，第i列的取值数为$Nkey_i$
  - 那么分组数 <= $Nkey_1$*$Nkey_2$ * … * $Nkey_k$

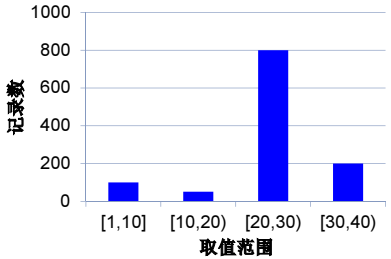
- 估计为0.5 N_{record}和$Nkey_1$*$Nkey_2$ * … * $Nkey_k$的较小值

改进的方法：采用统计直方图



等宽直方图：每个桶的取值范围宽度相同
上述为等宽直方图
等深直方图：每个桶的记录数相同

例如：估计col >25的记录数



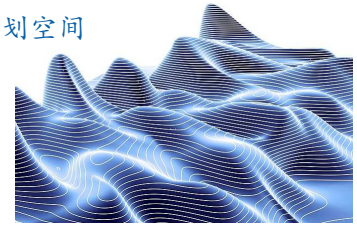
- 包含[30,40)的桶：200
- 在[20,30)的桶中：认为均匀分布uniform distribution
  - 那么：800* (30-25)/(30-20) = 400
- 共：200+400=600

Outline

- 查询优化概述
- 将SQL查询转换成关系代数表达式
- 关系代数的等价变换
- 运算代价的估计
- 基于成本的计划选择
- 多个查询块：嵌套子查询、视图、集合运算

Cost-Based Plan Selection

- 优化目标
  - min cost(物理执行计划)
- 限制条件：
  - 物理执行计划是给定逻辑执行计划的一个可行的实现
- 搜索可行的物理执行计划空间
- 寻找最优/较优解



穷举法

- 通过某种方式列举所有可能物理计划
- 对每个计划进行代价估计
- 找出最小代价计划

- 优点：可以找出最优计划
- 问题：组合爆炸！
- 适合物理计划数少的查询

启发式方法（Heuristic Methods）

- 根据事先定义的启发式规则，选择执行计划
- 例如
  - 采用局部的优化：贪心算法
    - 分别确定每个数据表的数据访问路径：扫描？索引？
    - 采用贪心算法来寻找连接的顺序
  - 定义简单的规则
    - 如果连接的一个输入表在连接属性上有索引，那么采用nested loop index join
    - 如果连接的一个数据表是排序的，那么采用sort merge join
    - 要计算三个或多个关系的交集时，先对最小的关系求交

分支定界法（Branch-and-Bound）

1. 采用启发式方法找到一个较好的物理计划
2. 设当前已知的计划中最优的代价为C
3. 通过某种系统的方法产生对全部搜索空间的遍历
   - 在遍历中，对搜索的局部空间进行代价的下界估计
   - 如果发现下界高于C，那么就可以剪枝这个局部空间
   - 每找到一个可行计划，都进行代价估计，如果代价小于C，那么更新目前的最优计划和代价

爬山法（Hill Climbing）

- 以一个启发式产生的计划为初始计划
- 对于每个计划考虑所有可能的变换
  - 寻找减少代价最有效的变换，产生新的计划
- 不断执行上述步骤，直至无法优化为止

- 模拟退火（Simulated Annealing）
  - 爬山法和随机跳跃结合
  - 爬山找到局部最优，然后试图随机跳跃，找到其它点

动态规划（Dynamic Programming）

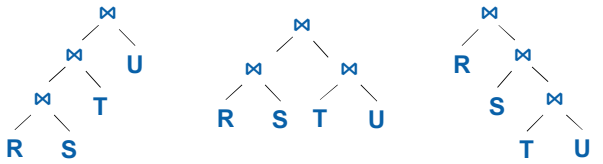
- 自底向上对执行计划树产生物理计划
- 在一棵树的最优计划里，所有子树的计划也是最优的
- 利用这一特点，减少穷举的空间

考虑一个具体问题：连接顺序的选择

- 两个关系的连接$R \bowtie_{R.a=S.b} S$
  - 对于一个具体的算法，例如hash join，nested loop
  - R是第一个表，还是S是第一个表？
  - 代价是不同的

- k个关系的连接$R_1 \bowtie R_2 \bowtie \ldots \bowtie R_k$
  - 有k!种顺序

确定了顺序后，还可以有不同种连接方案

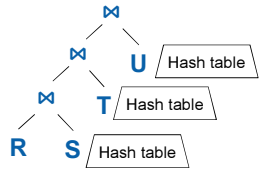


于是

- left-deep plan的个数？
  - k!
- right-deep plan的个数？
  - k!
- 允许所有可能结构的计划个数？
  - T(n) = $\sum_{i=1}^{n-1} T(i)T(n-i)$

Left-Deep计划

- 很多系统只考虑left-deep计划
  - 减少了搜索空间
  - 假设对于hash join，左关系为probe，右关系为build
    - 对于内存较大的情况，S, T, U都可以建立内存哈希表
    - 然后扫描一遍R，就可以完成join



连接顺序：
动态规划算法（Dynamic Programming）

- 数据结构中每一项
  - 多个关系、这些关系上的最佳计划、最佳计划的代价

初始化：对所有单个关系添加一项，代价为0；

for (j=2; j<=k; j++) {

对于所有j个关系的组合添加一项，
考虑所有j-1计划，计算最佳计划和代价；

}

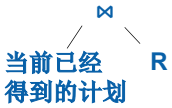
对于j个关系的组合计算最佳计划

- 如果只考虑left-deep plan
- 那么，对于每个关系考虑下述形式

- 这样得到j个不同的计划，选择其中代价最小的，就是最佳计划

连接顺序：贪心算法

1. 初始化：选择最小的关系表
2. 考虑每个还没有使用的关系，找到使下一步连接代价最小的R



3. 把R添加到当前计划中
4. 重复步骤2，直至所有关系都考虑了

Outline

- 查询优化概述
- 将SQL查询转换成关系代数表达式
- 关系代数的等价变换
- 运算代价的估计
- 基于成本的计划选择
- 多个查询块：嵌套子查询、视图、集合运算

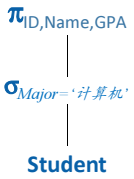
视图View

- create view：
  - 记录了view的定义
  - 可能在内部转化为关系代数树的形式
- SQL语句使用view时
  - 把关系代数树与SQL语句本身进行结合
  - 然后，进行关系代数的变形转化
- 情况1
  - View可以与查询合并成为一个SQL块
- 情况2
  - View需要作为一个子查询来实现

举例

- 假设我们经常要查询计算机系学生姓名和所选课程

```
create view CSStudent as
    select ID, Name, GPA
    from   Student
    where Major='计算机';
```



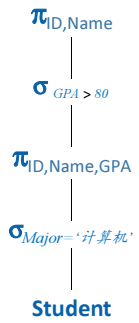
举例

select *ID*, *Name*
from *CSStudent*
where *GPA* > *80*;

把SQL表达为关系代数

举例

select *ID*, *Name*
from *CSStudent*
where *GPA* > *80*;

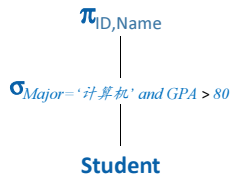


把View定义代入

举例

select *ID*, *Name*
from *CSStudent*
where *GPA* > *80*;

关系代数等价变形：
这里成为了一个SQL块



嵌套子查询

- 情况一：子查询与主查询无关
  - 那么可以先计算子查询
  - 然后在子查询结果的基础上，计算主查询

```
select S.sname
from Sailors S
where S.rating = (select MAX(S2.rating)
                  from Sailors S2);
```

嵌套子查询

- 情况二：有些子查询可以转化为连接

```
select S.sname
from Sailors S
where S.sid in (select R.sid
                from Reserves R
                where R.bid = 103);
```

- 这里子查询是与主查询无关的，所以可以先算出来
- 或者，可以把它转化为S与R的连接操作

嵌套子查询

- 情况二：有些子查询可以转化为连接

```
select S.sname
from Sailors S
where exists (select R.sid
              from Reserves R
              where R.bid = 103 and S.sid=R.sid);
```

- 子查询与主查询是相关的
- 可以把它转化为S与R的连接操作

嵌套子查询

- 除了有限的情况外，大部分只能通过嵌套循环计算
- 对于主查询的每个记录，计算一次子查询
- 所以，嵌套查询的代价有可能很大

集合运算

```
(select  sname
from Sailors
where age > 20)

except

(select  sname
from Sailors, Reserves
where Sailors.sid=Reserves.sid
);
```

计算每个集合的SQL块

如果有多个集合运算，需要寻找最佳的求解顺序

- 与连接处理类似

小结

- 查询优化概述
- 将SQL查询转换成关系代数表达式
- 关系代数的等价变换
- 运算代价的估计
- 基于成本的计划选择
- 多个查询块：嵌套子查询、视图、集合运算