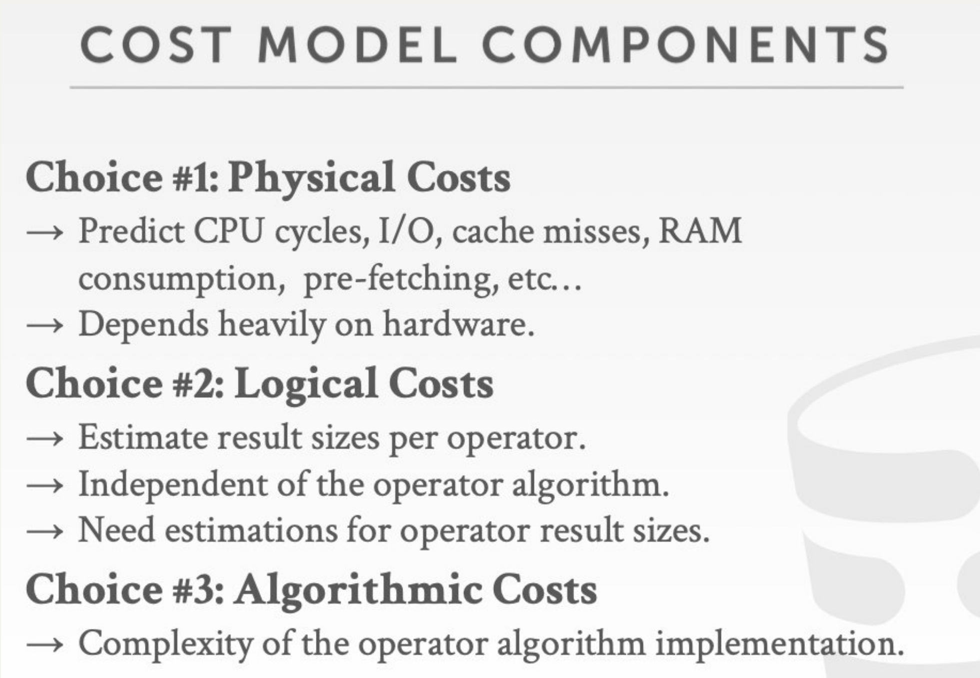
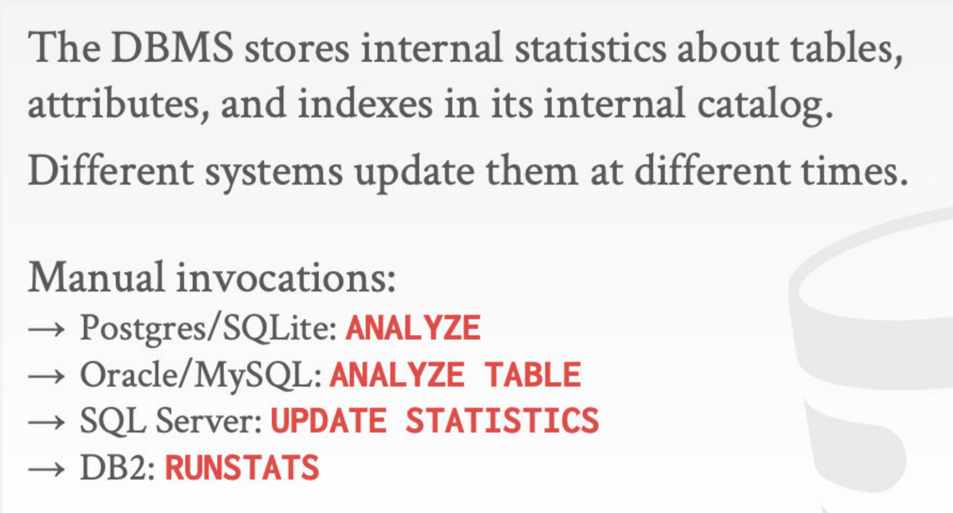
本Lecture继续介绍基于代价（cost-based）的查询优化器

Moe Cost Estimation (Statistics)



上一个Lecture中最后提到的代价模型的三部分（上图）都依赖于DBMS内部的统计信息（比如说算子要处理的数据的规模），可以通过如下所示的命令手动更新DBMS的统计信息，并且DBMS也会周期性地自动更新统计信息



DBMS的统计信息当中，有如下的一些概念

For each relation R, the DBMS maintains the following information:

→ N_R: Number of tuples in R.

→ V(A,R): Number of distinct values for attribute A.

The **selection cardinality** SC(A,R) is the average number of records with a value for an attribute A given N_R / V(A,R)

Note that this formula assumes ***data uniformity*** where every value has the same frequency as all other values.

→ Example: 10,000 students, 10 colleges – how many students in SCS?

可以看到，`SC(A,R)`是在数据均匀分布的假设之下的概念

基于这些概念，便有了如下的logical costs中的基数计算方法：

Equality predicates on unique keys are easy to estimate.

```
SELECT * FROM people
 WHERE id = 123
```

```
CREATE TABLE people (
  id INT PRIMARY KEY,
  val INT NOT NULL,
  age INT NOT NULL,
  status VARCHAR(16)
);
```

Computing the logical cost of complex predicates is more difficult...

```
SELECT * FROM people
 WHERE val > 1000
```

```
SELECT * FROM people
 WHERE age = 30
   AND status = 'Lit'
   AND age+id IN (1,2,3)
```

·DB
Fall 2021)

如果数据库的表中的某列所有字段都是unique的（比如说主键所在列），那么关于这一列的equality谓词能筛选出的tuple数量只能是1或0

对于其他情况下的复杂谓词来说，为了计算出它们的相关基数与开销，就需要引入选择率（selectivity）这样一个概念，形象的说，就是“谓词能选上来百分之多少的数据”，

The **selectivity** (sel) of a predicate P is the fraction of tuples that qualify.

Formula depends on type of predicate:

→ Equality
→ Range
→ Negation
→ Conjunction
→ Disjunction

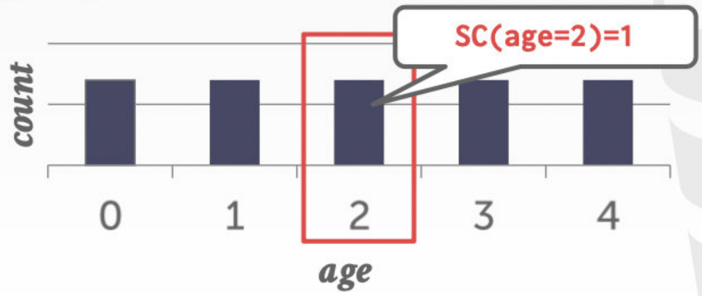
在前面的数据均匀分布的模型下，便有如下的计算谓词选择率的方法

Assume that V(age,people) has five distinct values (0–4) and N_R = 5

```
SELECT * FROM people
 WHERE age = 2
```

Equality Predicate: A=constant

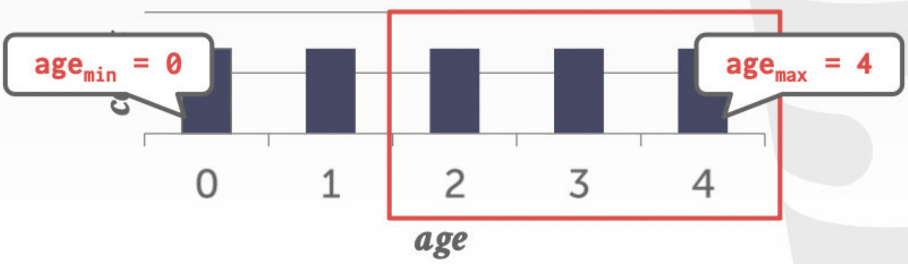
→ sel(A=constant) = SC(P) / N_R
→ Example: sel(age=2) = 1/5

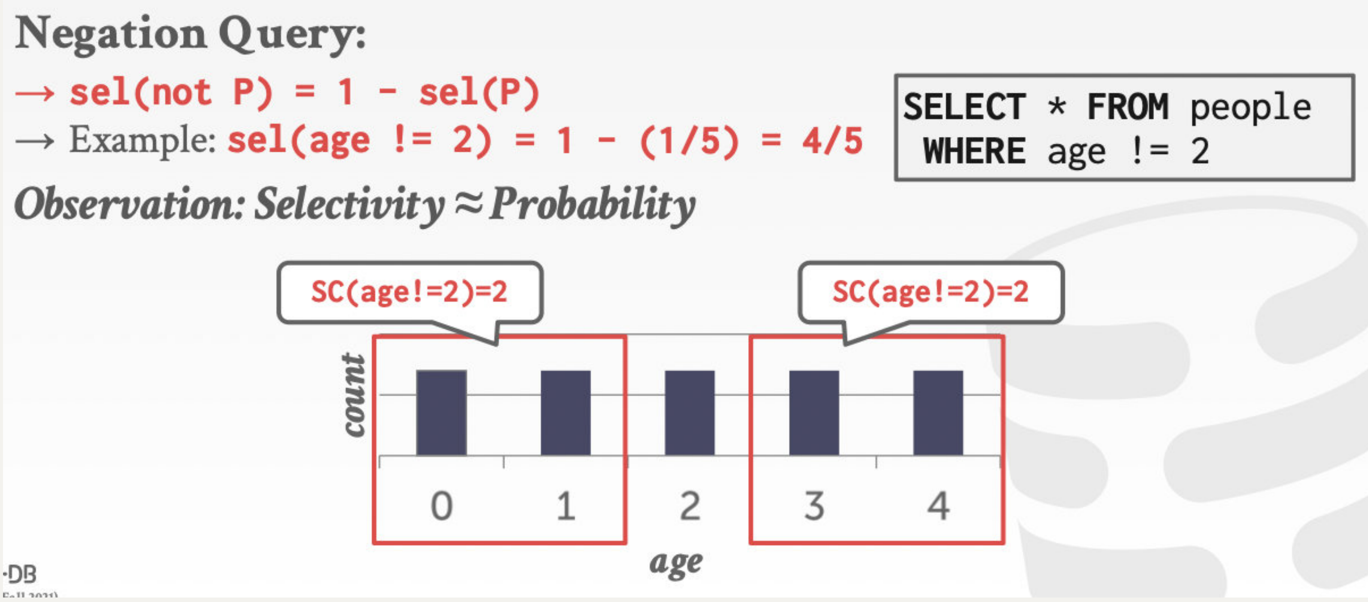


Range Predicate:

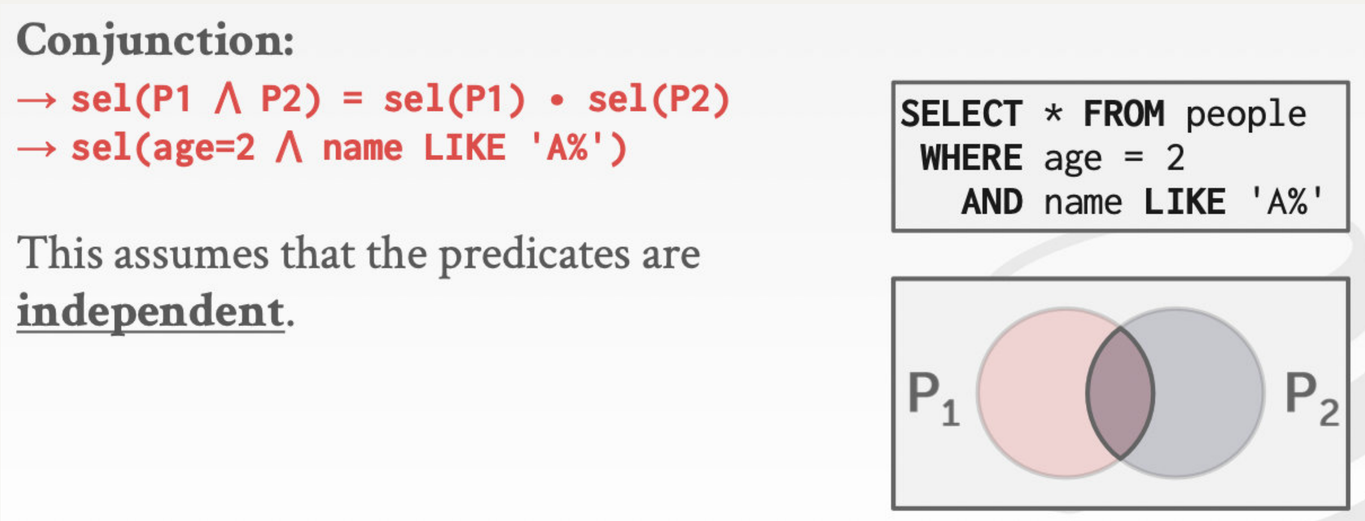
→ sel(A>=a) = (A_{max} - a+1) / (A_{max} - A_{min}+1)
→ Example: sel(age>=2) ≈ (4-2+1) / (4-0+1)
≈ 3/5

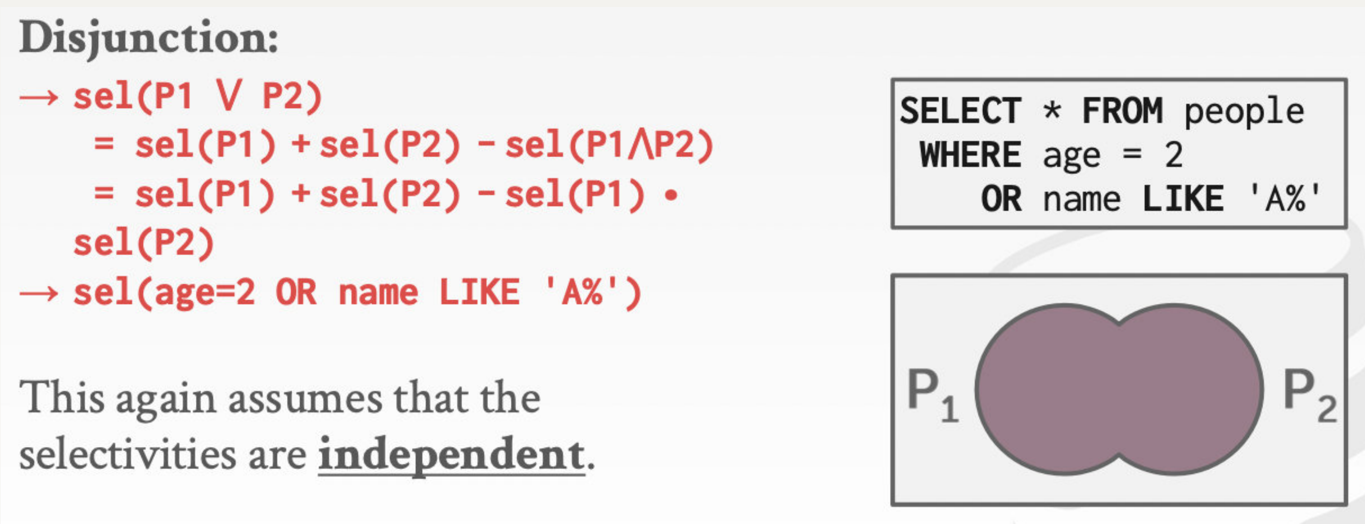
```
SELECT * FROM people
 WHERE age >= 2
```





可以察觉到：选择率和数据出现的概率是很相似的概念。多谓词情况下，可以使用计算概率的方法，根据每个谓词的选择率计算总体的选择率（如下所示）





前面所讨论的都是和select语句相关的基数的计算，对于带有join操作的语句来说，两个表join得到的结果集的规模该如何计算呢?

Given a join of **R** and **S**, what is the range of possible result sizes in # of tuples?

In other words, for a given tuple of **R**, how many tuples of **S** will it match?

Assume each key in the inner relation will exist in the outer table

基于上述假设（内表的每个join key都能在外表匹配到tuple），有如下的计算公式（本质上还是基于数据均匀分布的模型）

General case: $R_{cols} \cap S_{cols} = \{A\}$ where **A** is not a primary key for either table.

→ Match each **R**-tuple with **S**-tuples:

$estSize \approx N_R \cdot N_S / V(A,S)$

→ Symmetrically, for **S**:

$estSize \approx N_R \cdot N_S / V(A,R)$

Overall:

→ $estSize \approx N_R \cdot N_S / \max(\{V(A,S), V(A,R)\})$

总计一下，上述的公式都是基于如下的三个假设:

Assumption #1: Uniform Data

→ The distribution of values (except for the heavy hitters) is the same.

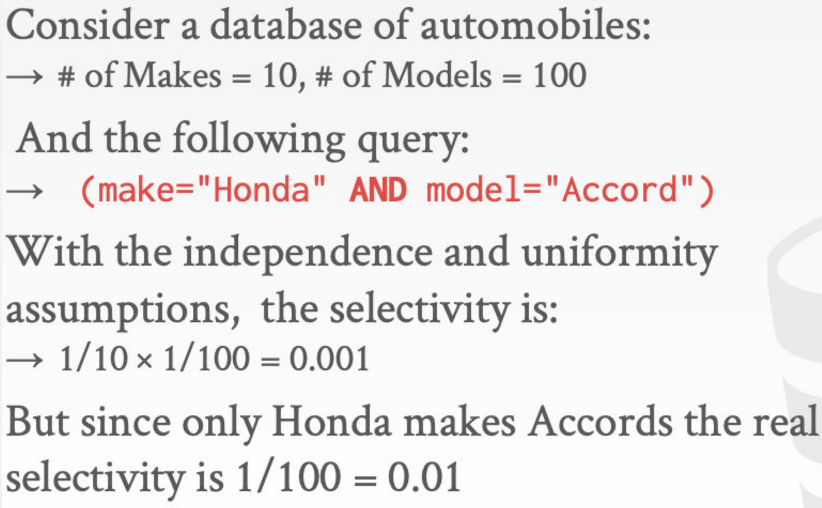
Assumption #2: Independent Predicates

→ The predicates on attributes are independent

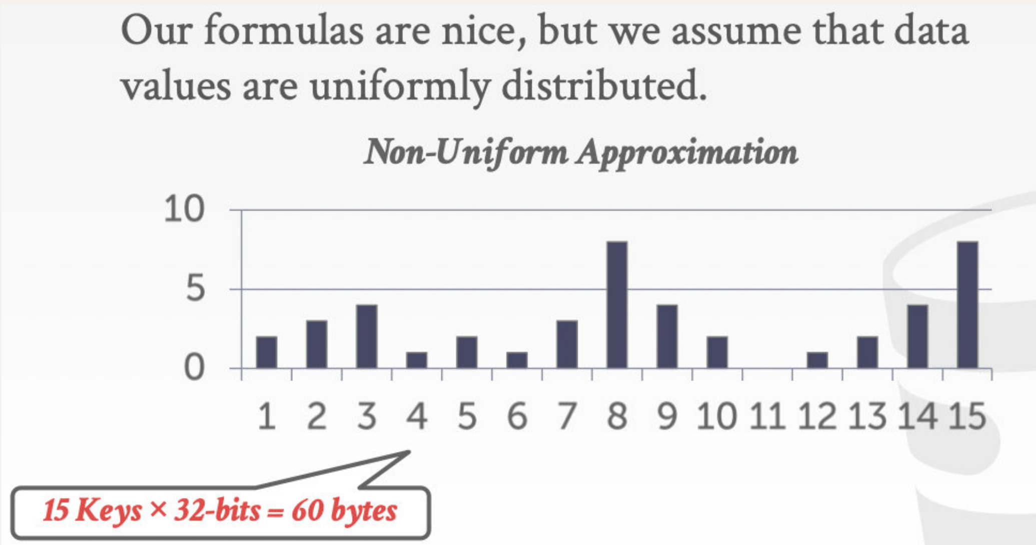
Assumption #3: Inclusion Principle

→ The domain of join keys overlap such that each key in the inner relation will also exist in the outer table.

实际应用场景并不会这么理想化，比如说：关于不同的attribute的不同谓词之间往往不是完全独立的，因此上面的基于独立事件概率模型计算谓词选择率的公式就会失效（如下所示）



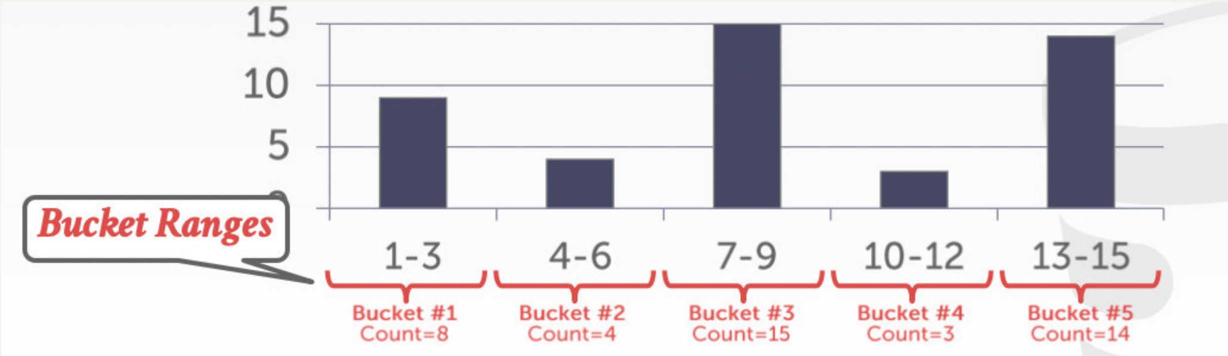
对于不均匀分布/attribute之间不独立的数据，我们该使用什么样的手段来对其进行统计呢？



针对不均匀分布的数据，可以对每个key都记录它对应多少个value（结合上面的图表来说，它对应的场景可以是：在数据库中记录的对象中，1~15岁的人各有多少。年龄是key，“每个key对应几个value”说的就是“有多少人是这个年龄的”），但这么做也有弊端，如果表很大的话，统计信息也会非常庞大，因此有如下改进方法：

- 使用等宽直方图

  把key所在的区间切分成等宽的小区间，每个小区间对应一个bucket，不再记录每个key对应着多少value，而是记录每个bucket对应着多少value（如下所示）

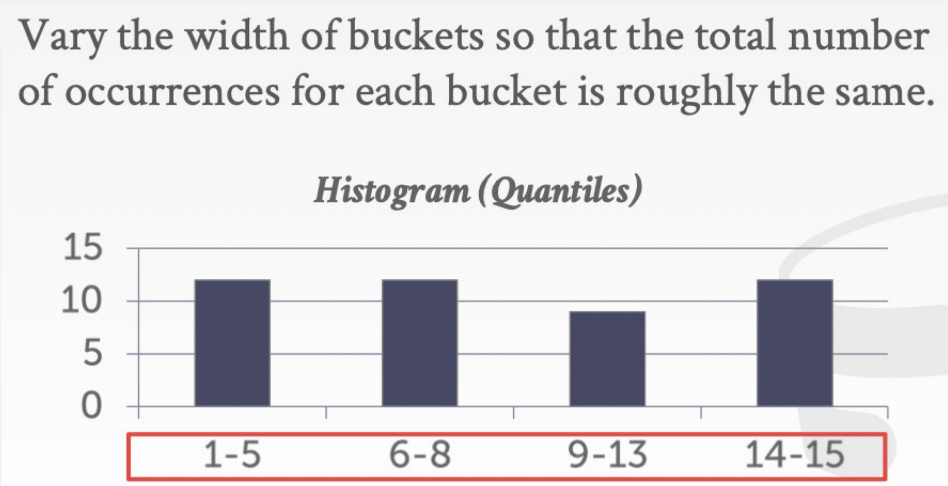


但这也有一些问题，在最初的直方图里，11这个key没有value与它对应，7和8这两个key所对应的value的数目差别较大，也就是说这组被统计的数据的方差不小。但DBMS却只能拿bucket对应多少个value来反推bucket中含有的key对应多少个value，这会导致信息丢失的比较多，误差变大

- 使用等深直方图

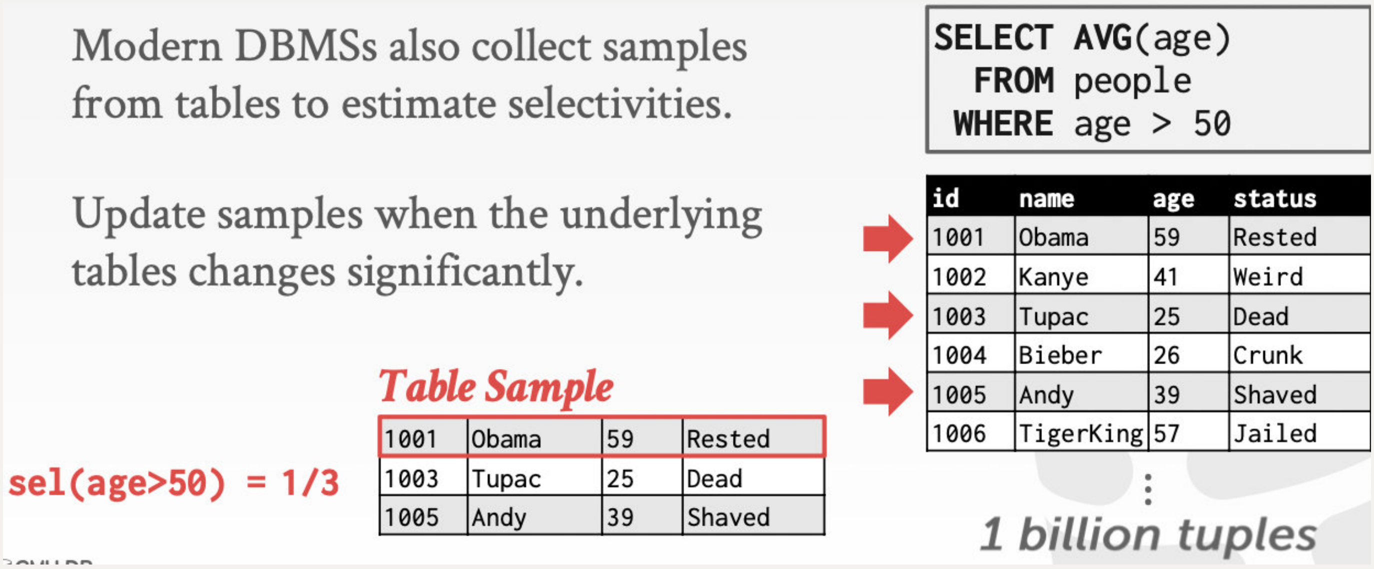
  这是基于等宽直方图的改进：每个bucket的宽度（即它含有几个key）不是固定的，但每个bucket里面的keys对应的value的数目是几乎一样的

  等深直方图节省了内存空间，又在一定程度上可以减少被统计数据的方差过大带来的误差



还有一种较为简单粗暴的统计方式，sampling-采样，这种策略的思想是：如果表特别大的话，我们不妨从其中随机选择一些tuple然后构成一个小表，把这个小表作为完整表的一个代表，然后下一步转而分析这个小表，将得出的统计信息用于对完整表的查询代价分析

举个例子说明（如下图所示），用户的SQL语句想筛选出数据库所记录的所有对象中大于50岁的人，DBMS优化器就可以对完整的表进行采样，然后分析小表，从而得出`age>50`这个谓词的选择率是1/3，那么我们便可以假设：在数据库的完整的表里面，`age>50`这个谓词的选择率也是1/3



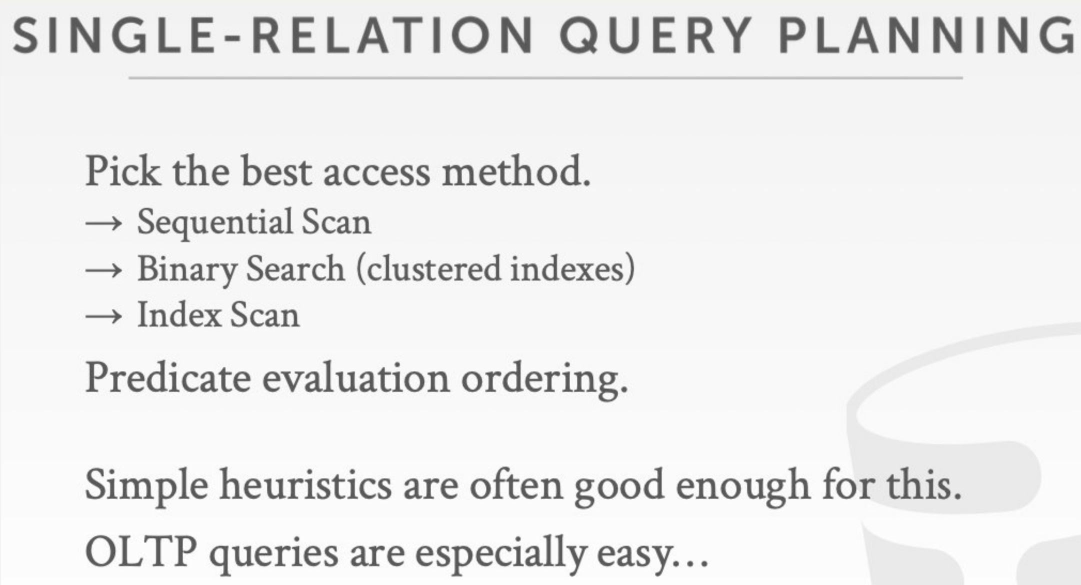
sampling也有一些问题：采样导致我们除了维护完整的表之后还要再额外维护小表（比如说采样时提取出来的tuple如果在完整的表里被删掉了，那么我们也要对小表做相应的改动），而且每个SQL语句在完整的表上面运行之前，还要先在小表上面跑一遍以获取统计信息，这毫无疑问会带来一定的开销

sampling的好处就是基于真实的数据去做估计

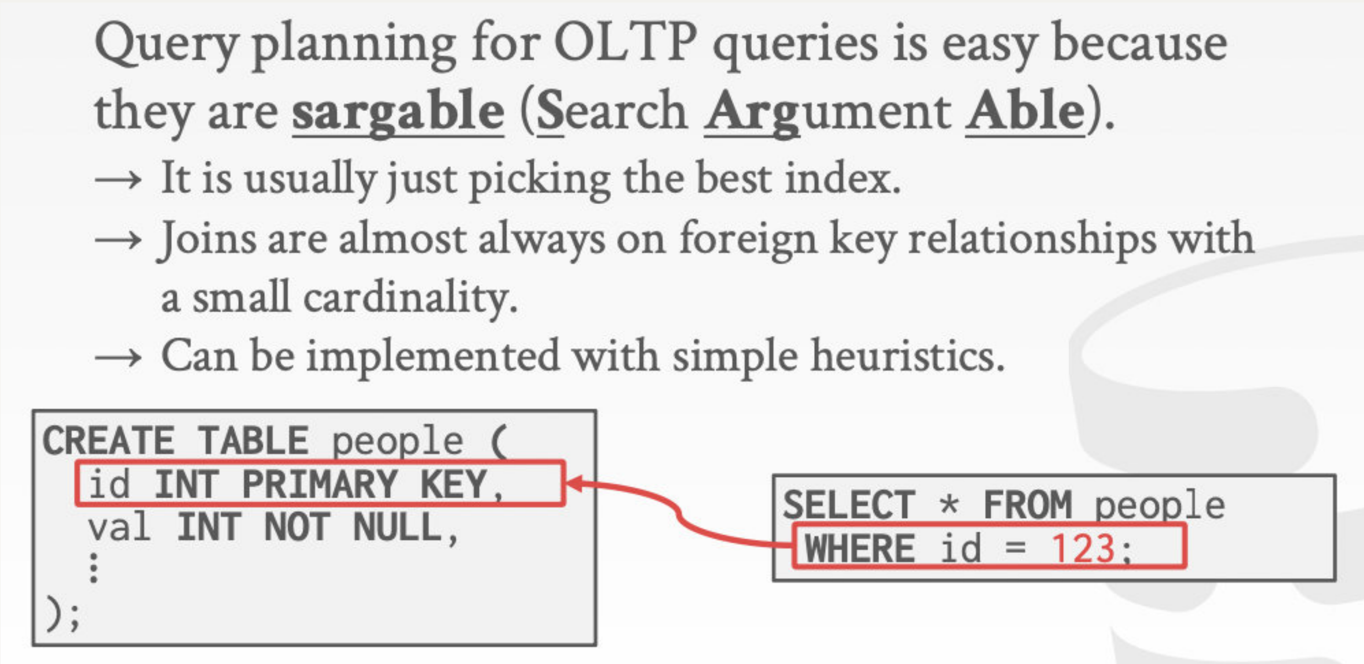
基于前面介绍的各种策略，我们可以粗略地去估计谓词的选择率（selectivity），在知道了谓词的选择率之后，就可以计算出有多少数据被送入了每个算子，知道了这个数据之后就可以计算每个算子的开销，进而得出整个执行计划的开销代价，知道了计划的开销，那么优化器就可以进入下一阶段：计划列举

Plan Enumeration

对于那些仅涉及单表的查询计划，优化器做的事情非常简单，可以仅仅启发式地依据规则对逻辑计划rewrite，比如说确定访问数据的最佳方式（相关的规则可以是：“若存在这个字段的索引，那么就走索引”），而不用去量化分析查询计划的开销



只涉及单表查询的OLTP型工作负载所对应的查询就可以使用上述策略做简单的优化，具体原因如下：

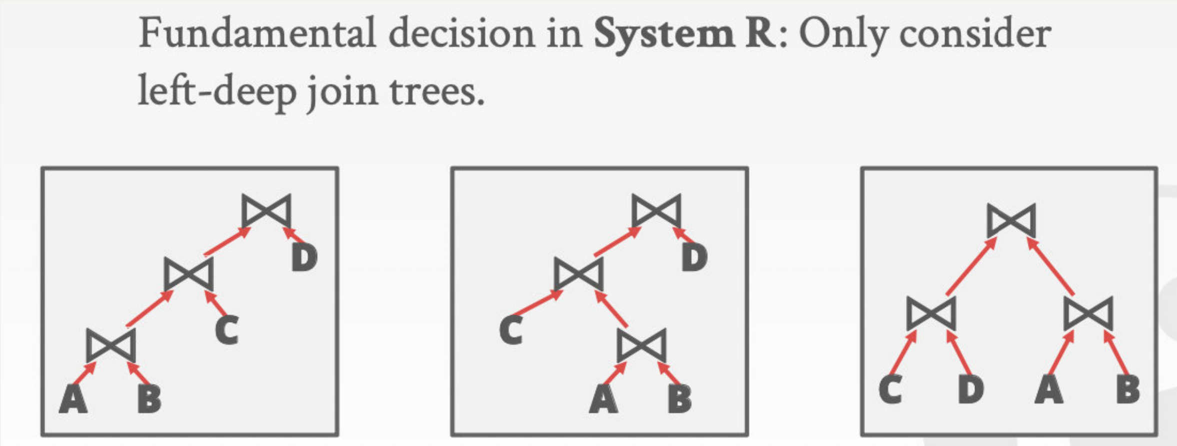


对于那些涉及到多个表的查询，不可避免地会有表与表之间的join，而inner-join运算既符合交换律，又符合结合律，因此4～5个表做join时计划列举的空间极大（join操作的顺序，join运算符左边和右边各是哪个表，这些都有极大的搜索空间，因此我们要用一些手段去降低这个搜索空间），IBM的System R便有了如下的规定

As number of joins increases, number of alternative plans grows rapidly
→ We need to restrict search space.

Fundamental decision in **System R**: only left-deep join trees are considered.
→ Modern DBMSs do not always make this assumption anymore.

System R只考虑左深树对应的join排列，即join的左子树也是一个join操作（如下图最左侧所示，因此下图的右边两个join排列都会被pass掉）

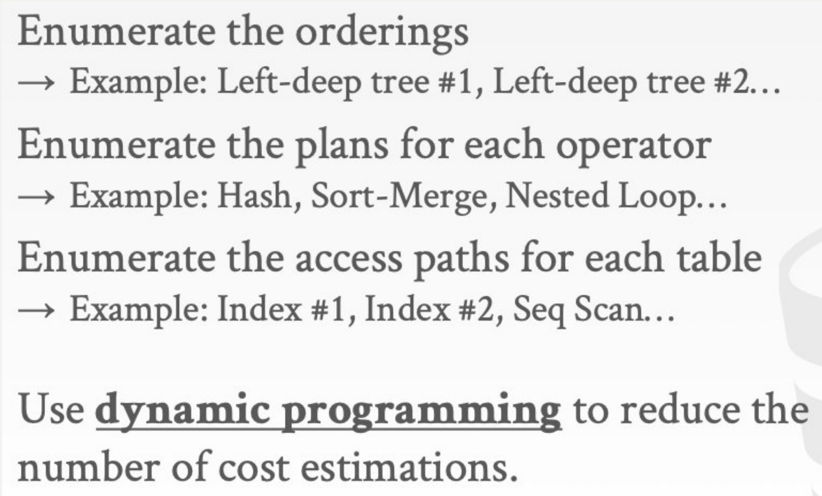


而且左深树带来了意想不到的好处：如果计划的执行模型（process model）是火山模型，那么就可以做到（假设B~D表的哈希表都做好了并且进行的是hash join）：A表和B表做join得到一个tuple，吐给上层的join算子，然后上层的join算子拿这个tuple和C表做join，之后再吐给上层，和D表做join，这便实现了几乎完美的流式操作，极大程度上使中间结果集更小

Fundamental decision in **System R** is to only consider left-deep join trees.

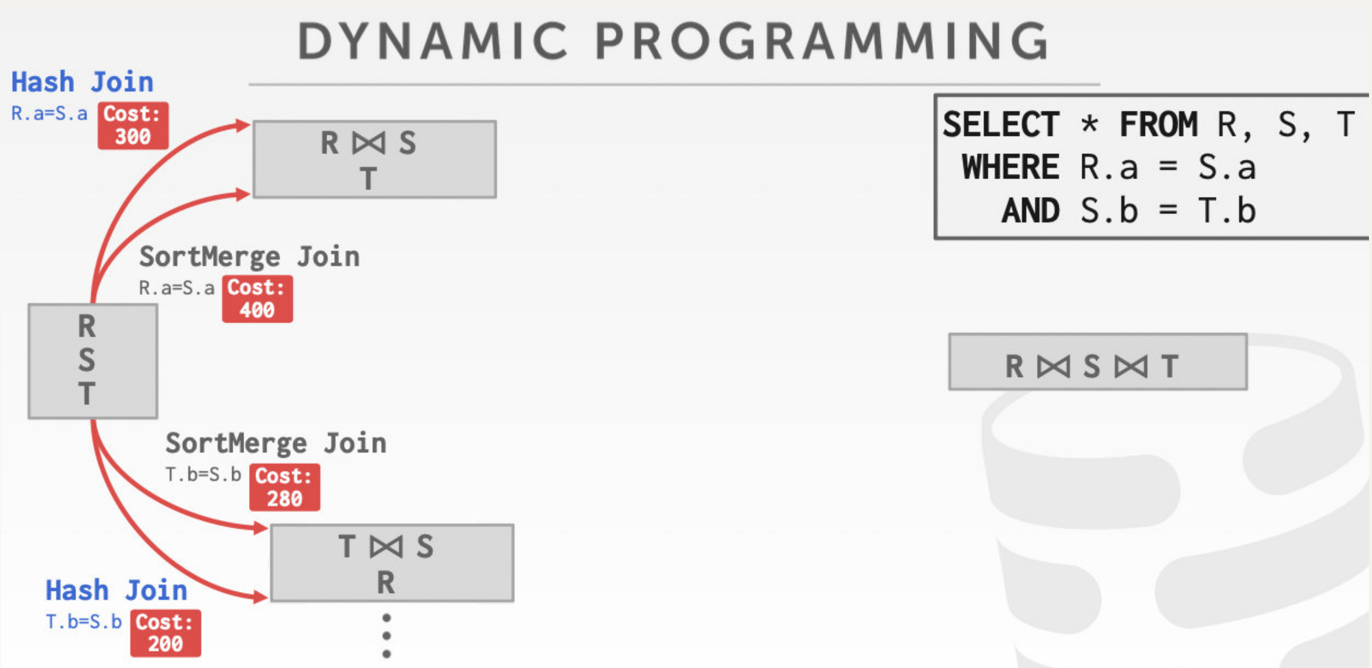
Allows for fully pipelined plans where intermediate results are not written to temp files.
→ Not all left-deep trees are fully pipelined.

除了System R，其他DBMS在多表查询时对查询计划的列举往往基于如下的三个方向：

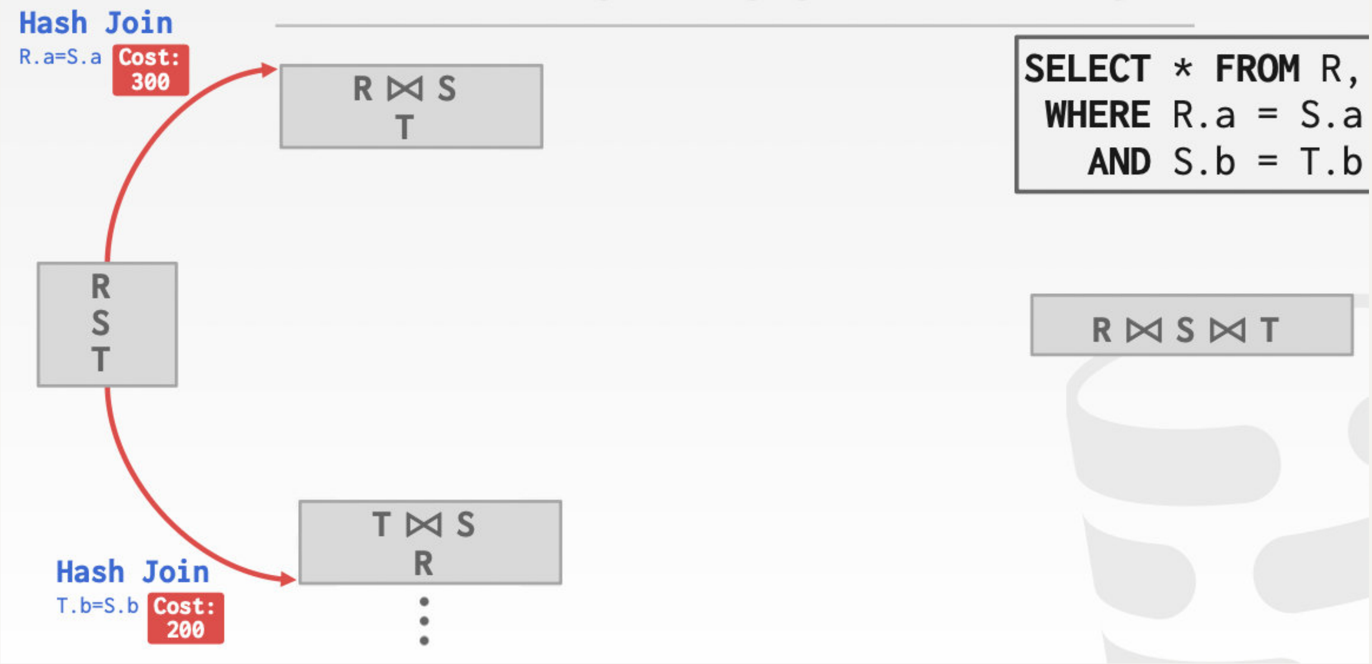


列举时往往使用动态规划去缩小搜索空间

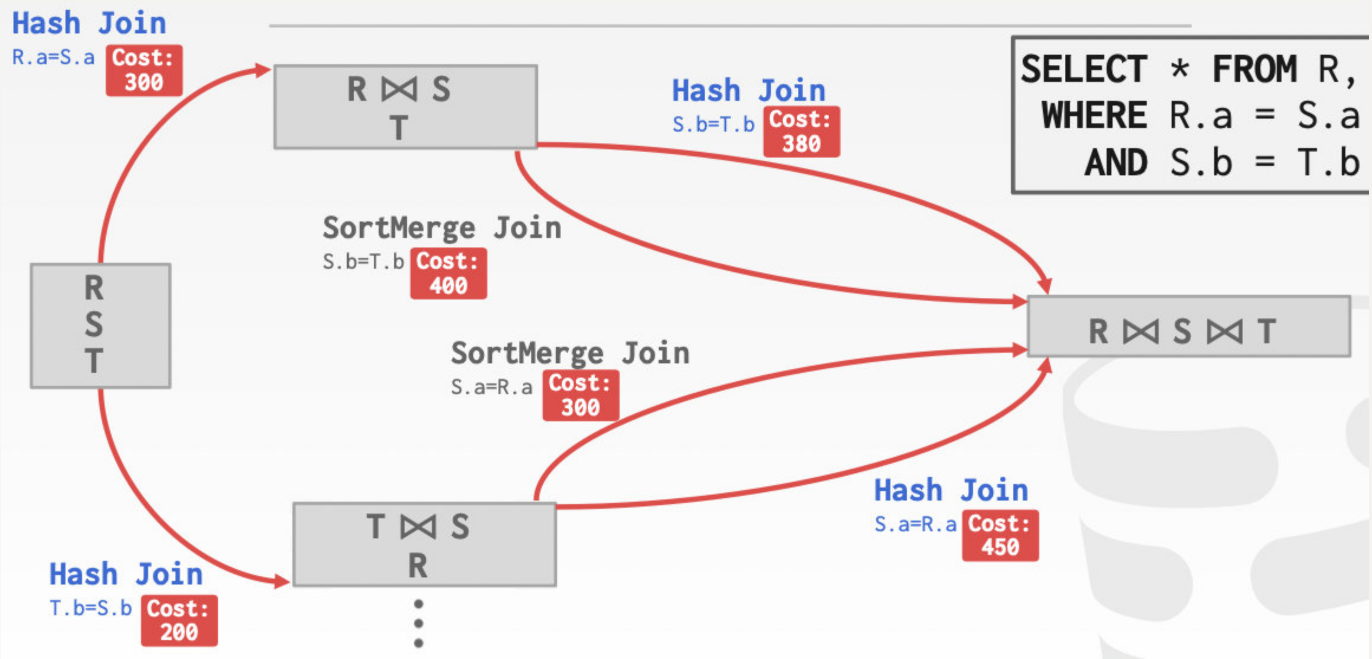
举个例子，R,S,T这三个表连表，那么就会有如下的状态机，并且DBMS可以计算出每条边对应的开销



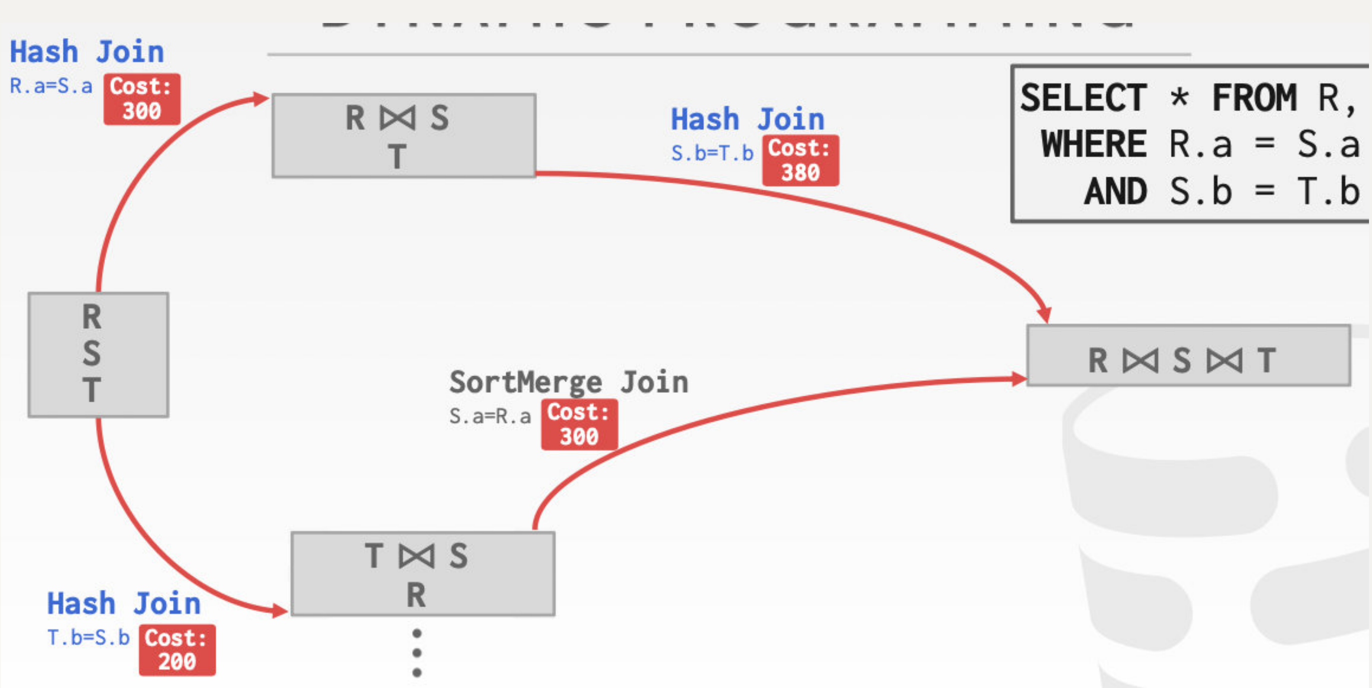
那么就可以做如下的剪枝，在重边里面舍弃开销大的边



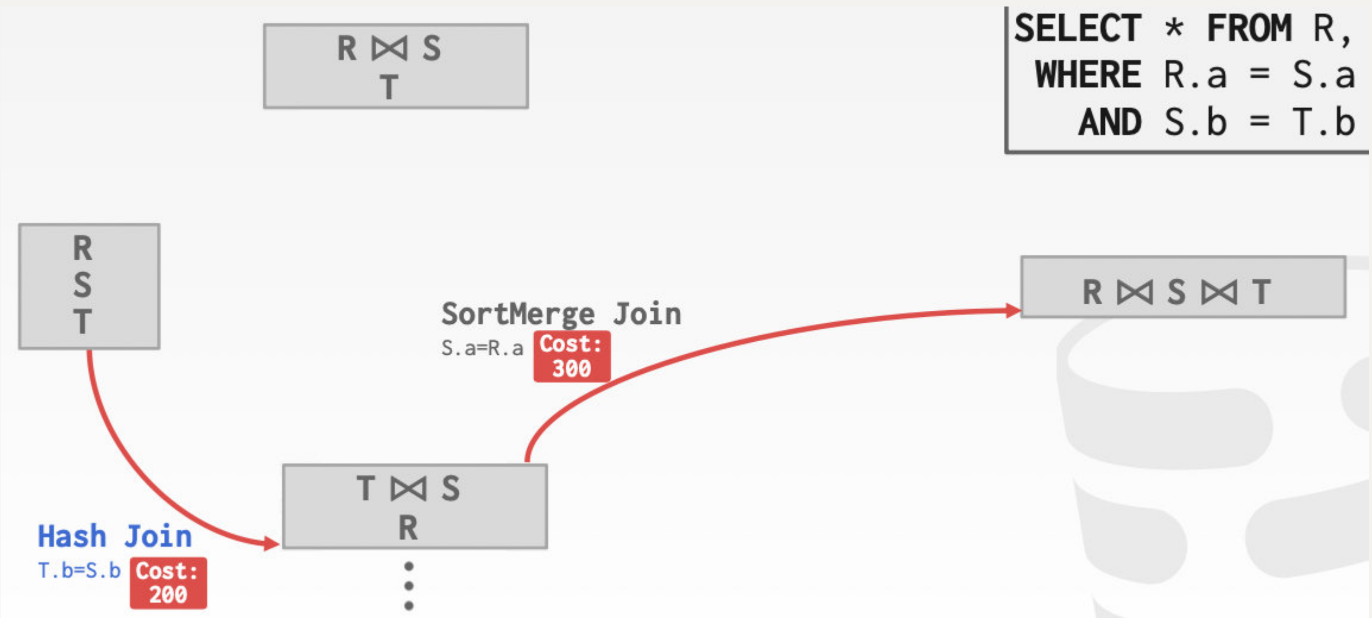
之后继续构建状态机，从中间状态走到终点



之后再次进行剪枝



最后再比较一下最后剩下的这两个路径的总开销，取较小的那个，这便是最优路径，其中包括并确定了join的顺序和每次join所采用的物理算子（即join的具体实现方法，使用哪种join算法）

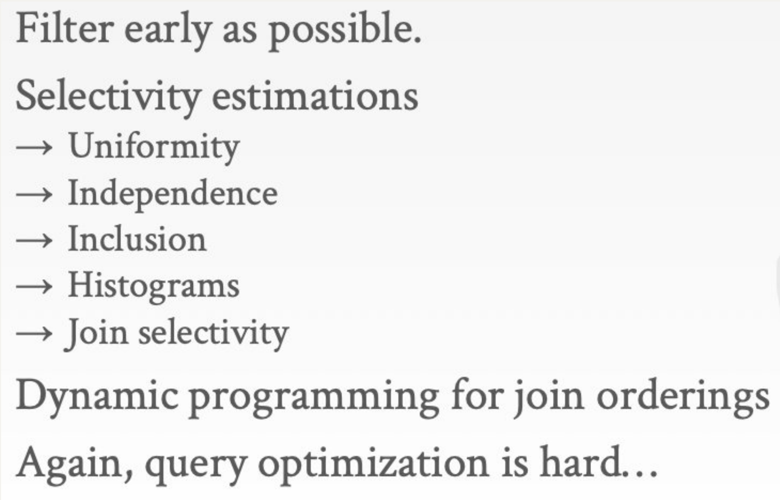


但上图只是一个简化后的计划列举，实际上还要考虑R表和S表要怎么读，走索引还是走扫描，读入内存的时候带不带顺序（这会影响物理算子的开销），以及一些其他的细节

当然，计划的列举也可以暴力枚举/暴力搜索，详见slides中的Candidate Plan Example

对于工业界的设计方案来说，Postgres数据库的优化器不仅采用了上面说的动态规划去进行计划列举，还采用了遗传算法这样的高级算法（详见slides）：当查询涉及的表较多时，就进行遗传算法（限定迭代的次数或时间），表较少时进行动态规划

最后总结一下查询优化，如下所示：



不要忘了，在启发式的查询优化中，应当尽早地进行数据的过滤（filter），谓词/算子的下推可以做到这一点

Ref/参考自：

https://www.bilibili.com/video/BV1qm4y197BQ/?spm_id_from=333.788