

CSC 261/461 – Database Systems

Lecture 22

Fall 2017

Announcement

- Quiz #9 (Free Quiz)
 - Due: This Wednesday
 - <https://goo.gl/forms/rkbwYw7uocLHKy5sI>
- Poster Printing Guidelines:
 - <https://piazza.com/class/j6it7h5r4gy3ti?cid=122>

Topics for Today

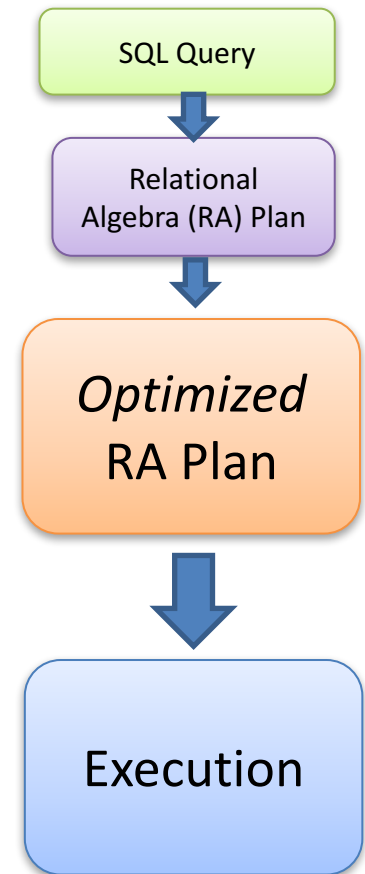
- Query Optimization (Chapter 19)

We will cover

1. Logical Optimization
2. Physical Optimization

Logical vs. Physical Optimization

- **Logical optimization:**
 - Find equivalent plans that are more efficient
 - Intuition: **Minimize # of tuples** at each step by changing the order of RA operators
- **Physical optimization:**
 - Find algorithm with **lowest IO cost** to execute our plan
 - Intuition: Calculate based on physical parameters (buffer size, etc.) and estimates of data size (histograms)



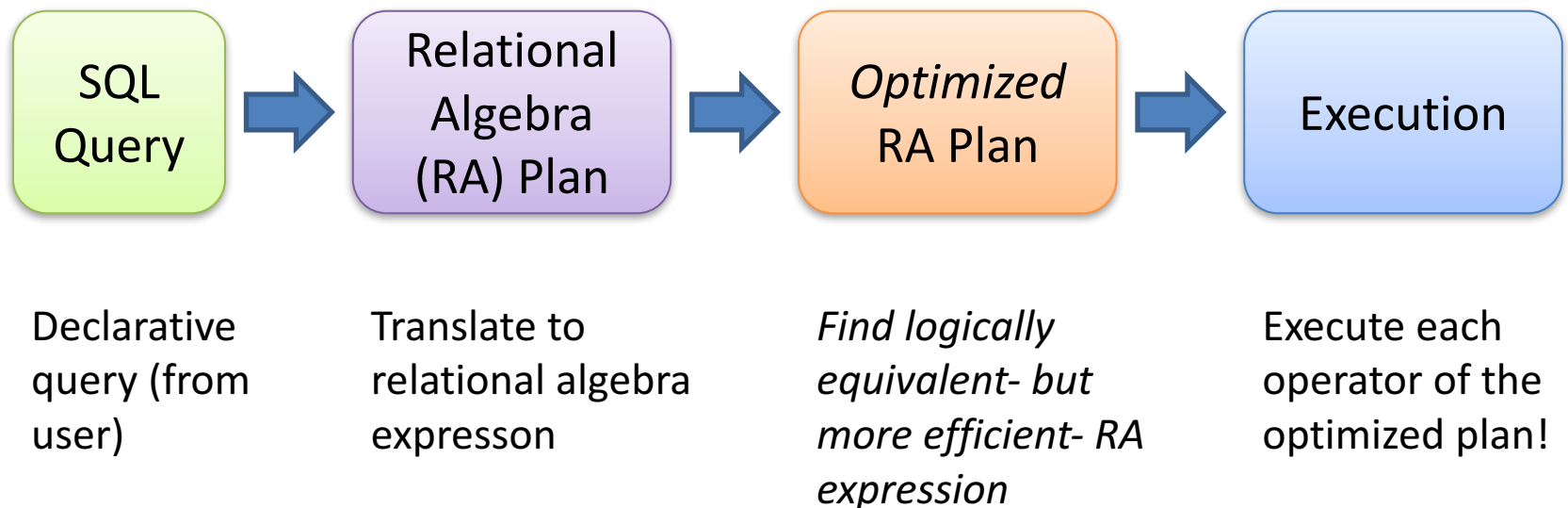
1. LOGICAL OPTIMIZATION

What you will learn about in this section

1. Optimization of RA Plans

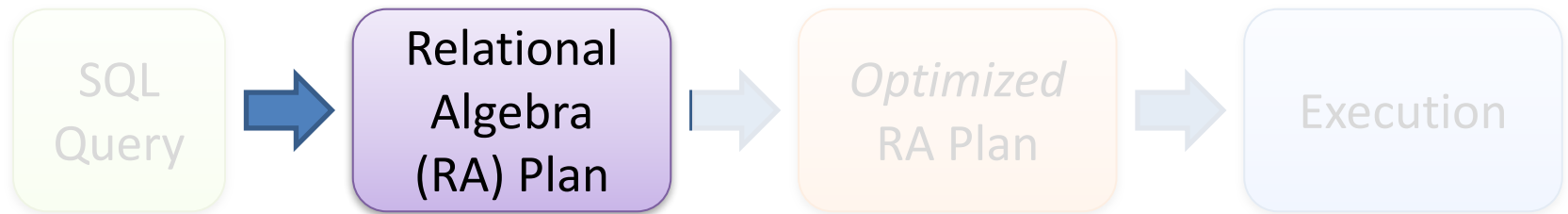
RDBMS Architecture

How does a SQL engine work ?



RDBMS Architecture

How does a SQL engine work ?



Relational Algebra allows us to translate declarative (SQL) queries into precise and optimizable expressions!

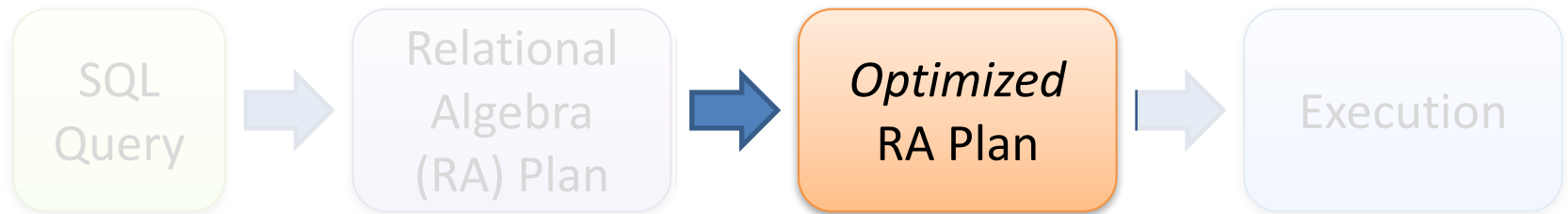
Recall: Logical Equivalence of RA Plans

- Given relations $R(A,B)$ and $S(B,C)$:
 - Here, projection & selection commute:
 - $\sigma_{A=5}(\Pi_A(R)) = \Pi_A(\sigma_{A=5}(R))$
 - What about here?
 - $\sigma_{A=5}(\Pi_B(R)) \neq \Pi_B(\sigma_{A=5}(R))$

We'll look at this in more depth later in the lecture...

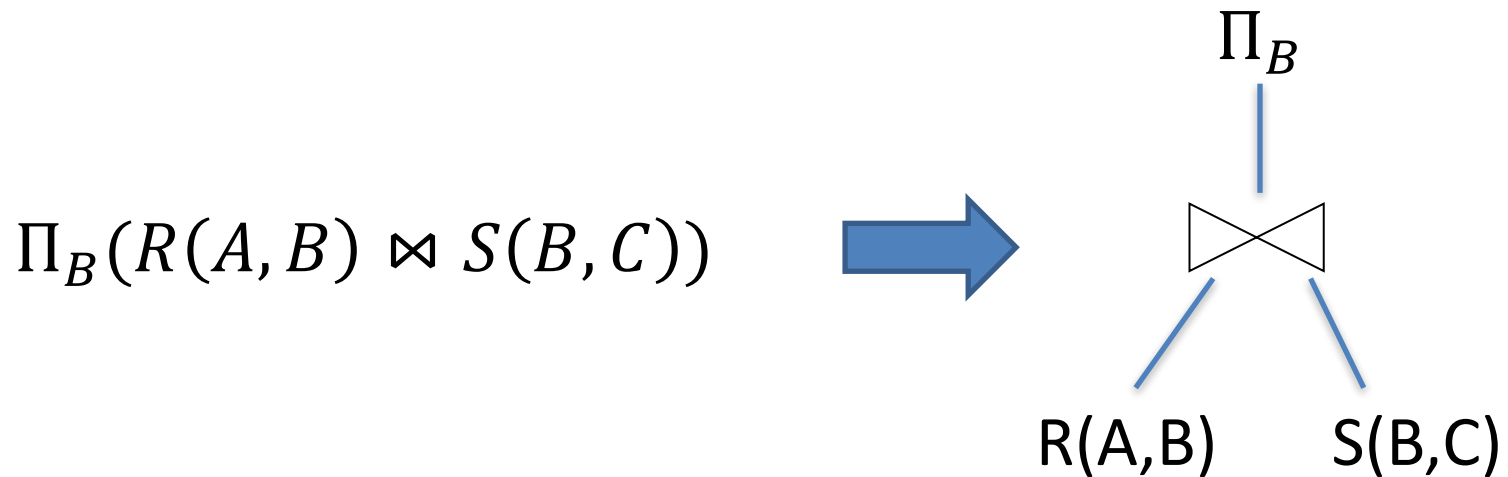
RDBMS Architecture

How does a SQL engine work ?



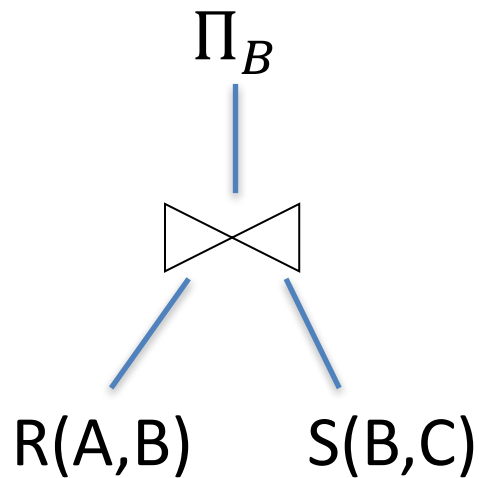
We'll look at how to then optimize these plans now

Note: We can visualize the plan as a tree



Bottom-up tree traversal = order of operation execution!

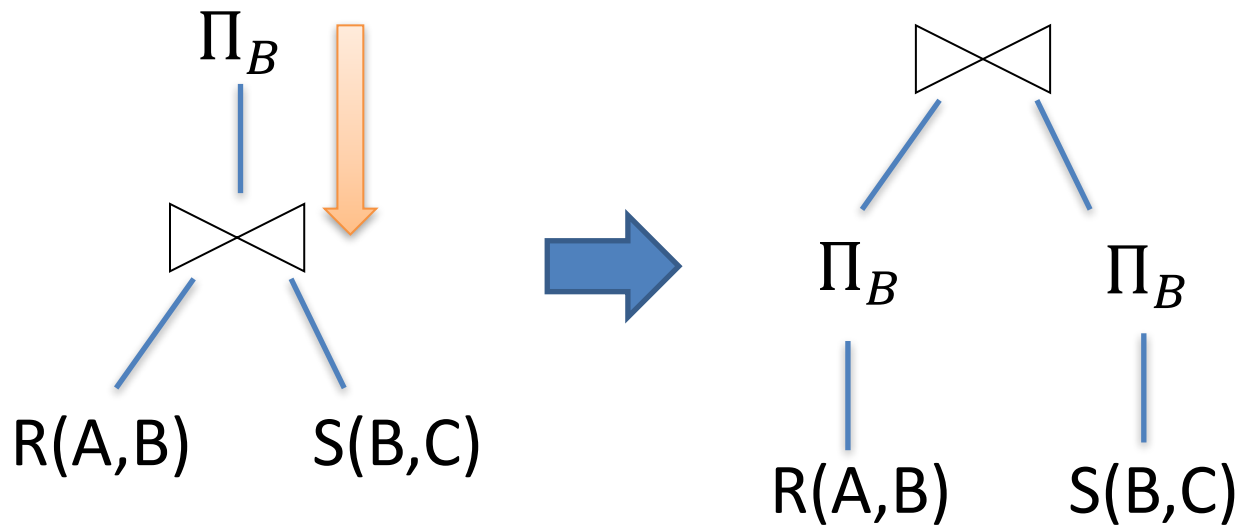
A simple plan



What SQL query does this correspond to?

Are there any logically equivalent RA expressions?

“Pushing down” projection



Why might we prefer this plan?

Takeaways

- This process is called **logical optimization**
- Many equivalent plans used to search for “good plans”
- Relational algebra is an important abstraction.

Optimizing the SFW RA Plan

RA commutators

- The basic commutators:
 - Push **projection** through (1) selection, (2) join
 - Push **selection** through (3) selection, (4) projection, (5) join
 - *Also*: Joins can be re-ordered!
- Note that this is not an exhaustive set of operations

This simple set of tools allows us to greatly improve the execution time of queries by optimizing RA plans!

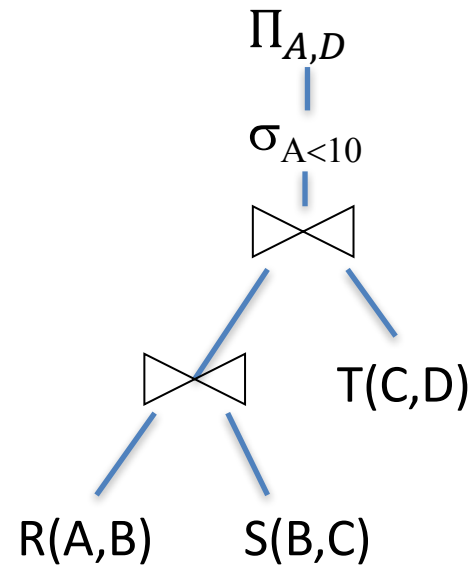
Translating to RA

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;



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



Logical Optimization

- Heuristically, we want selections and projections to occur as early as possible in the plan
 - Terminology:
 - “push down **selections**” and “push down **projections**.”
- **Intuition:** We will have fewer tuples in a plan.

Optimizing RA Plan

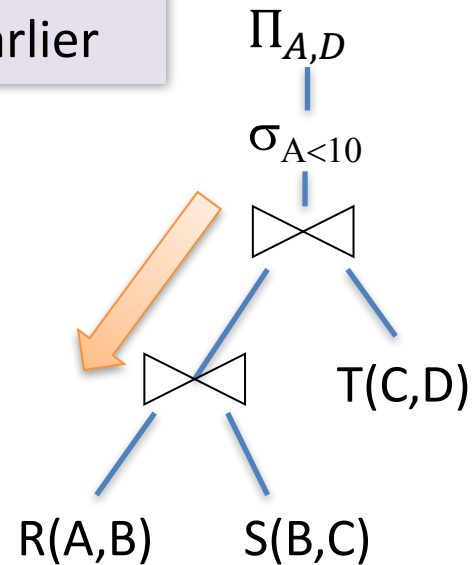
R(A,B) S(B,C) T(C,D)

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



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

Push down
selection on A so
it occurs earlier



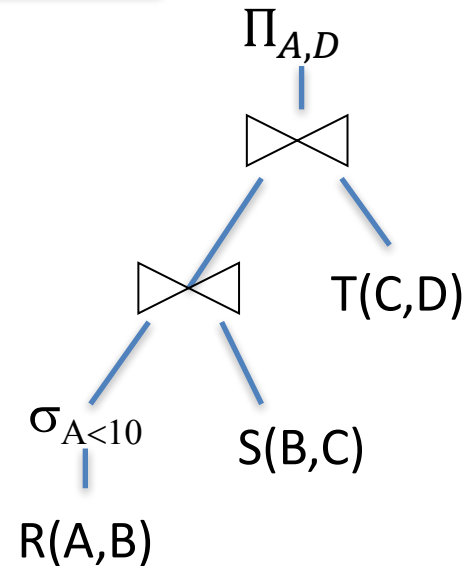
Optimizing RA Plan

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;
```


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

Push down
selection on A so
it occurs earlier



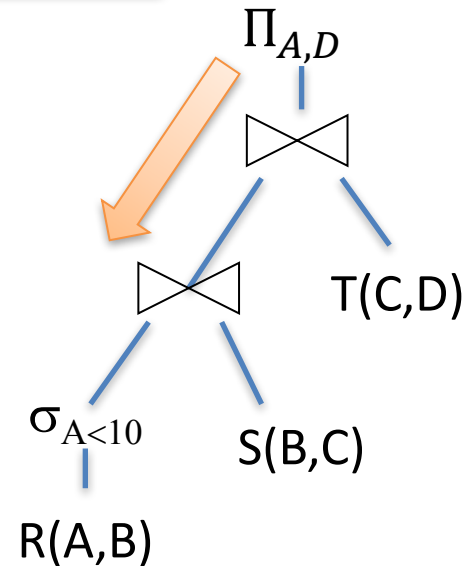
Optimizing RA Plan

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;
```


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

Push down
projection so it
occurs earlier



Optimizing RA Plan

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

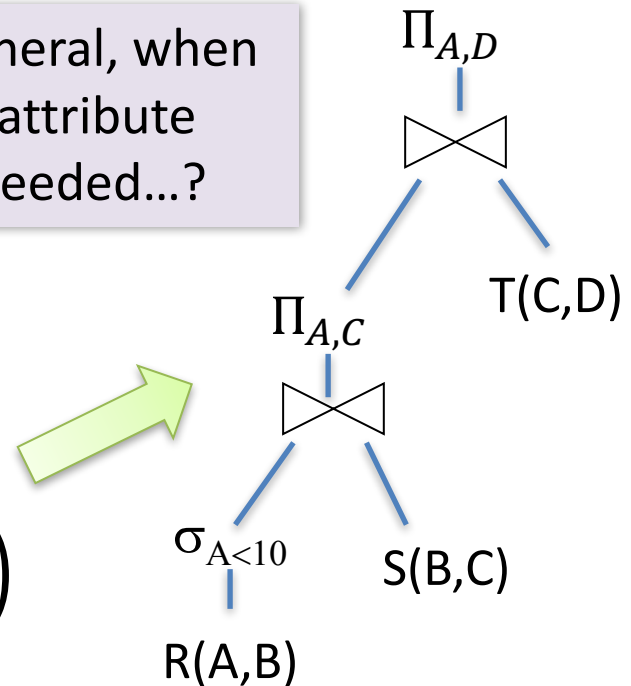
SELECT R.A, S.D
FROM R, S, T
WHERE R.B = S.B
AND S.C = T.C
AND R.A < 10;



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

We eliminate B
earlier!

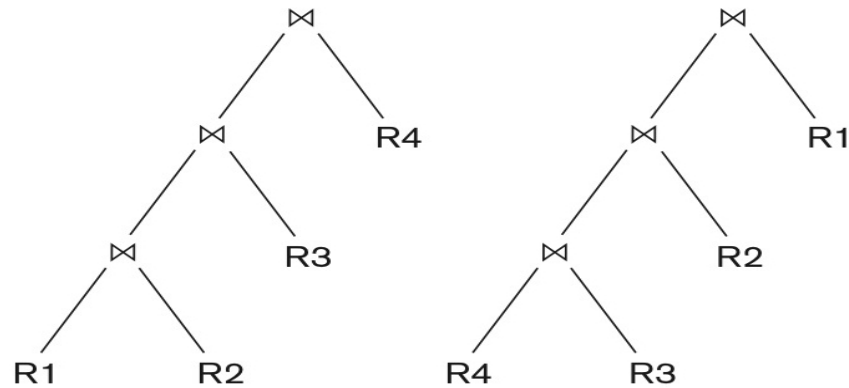
In general, when
is an attribute
not needed...?



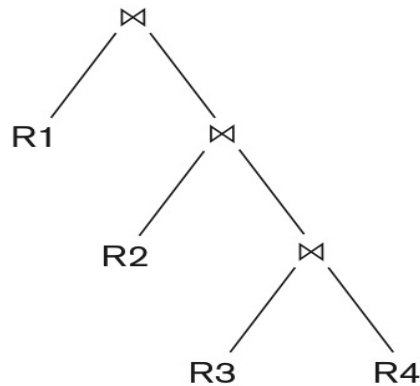
Logical optimization

- Selections and Cross Product can be **combined** into joins
- Selections and projections can be **pushed down** (below joins)
- Joins can be extensively **reordered**.

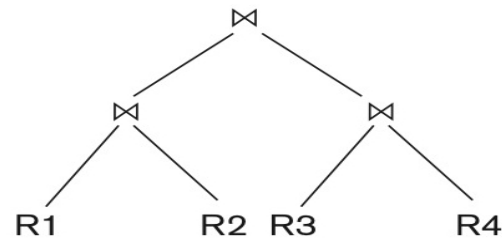
Equivalent Trees



(a)



(b)



(c)

(a) Two left-deep join query trees. (b) A right-deep join query tree. (c) A bushy query tree.

How to Handle Many Joins

Table 19.1 Number of Permutations of Left-Deep and Bushy Join Trees of n Relations

No. of Relations N	No. of Left-Deep Trees $N!$	No. of Bushy Shapes $S(N)$	No. of Bushy Trees $(2N - 2)! / (N - 1)!$
2	2	1	2
3	6	2	12
4	24	5	120
5	120	14	1,680
6	720	42	30,240
7	5,040	132	665,280

Reasons for Left-deep plans

- As the number of joins increases, the number of alternative plans increases rapidly. It becomes necessary to prune the space of alternative plans
- Left-deep trees allow us to **fully pipelined** plans.

2. PHYSICAL OPTIMIZATION

Cost functions for SELECT operation

- Terminology:
 - r : Number of records
 - b : Number of blocks
 - bfr : blocking factor
 - sl : selectivity (fraction of record satisfying the condition)
 - s : selection cardinality $= sl * r$
 - x : Number of levels (you can treat as depth)
 - l : number of leaves (# number of first level blocks)

Cost functions for Select (Section: 19.4)

Algorithm	Cost	Special Cases
Linear Search	$b/2$	b (if not found)
Binary Search	$\log_2 b + \left\lceil \left(\frac{s}{bfr} \right) \right\rceil - 1$	$\log_2 b$ (if on a unique key)
Primary Index	$x + 1$	
Hash Key	1	
Ordering index ($>$, $<$, $>=$, $<=$)	$x + (b/2)$	
Clustering Index	$x + \left\lceil \left(\frac{s}{bfr} \right) \right\rceil$	
B+ tree index	$x + 1 + s$	$x + l/2 + r/2$

What you will learn about in this section

1. Index Selection
2. Histograms

Index Selection

Input:

- Schema of the database
- **Workload description:** set of (query template, frequency) pairs

Goal: Select a set of indexes that minimize execution time of the workload.

- Cost / benefit balance: Each additional index may help with some queries, but requires updating

This is an optimization problem!

Example

Workload
description:

```
SELECT pname  
FROM Product  
WHERE year = ? AND category = ?
```

Frequency
10,000,000

```
SELECT pname  
FROM Product  
WHERE year = ? AND Category = ?  
AND manufacturer = ?
```

Frequency
10,000,000

Which indexes might we choose?

Example

Workload
description:

```
SELECT pname  
FROM Product  
WHERE year = ? AND category =?
```

Frequency
10,000,000

```
SELECT pname  
FROM Product  
WHERE year = ? AND Category =?  
AND manufacturer = ?
```

Frequency
100

Now which indexes might we choose? Worth keeping an index with manufacturer in its search key around?

Estimating index cost?

- Note that to frame as optimization problem, we first need an estimate of the *cost* of an index lookup
- Need to be able to estimate the costs of different indexes / index types...

We will see this mainly depends on getting estimates of result set size!

Ex: Clustered vs. Unclustered

Cost to do a range query for M entries over N -page file (P per page):

– Clustered:

- To traverse: $\log_f(1.5N)$
- To scan: 1 random IO + $\left\lceil \frac{M-1}{P} \right\rceil$ sequential IO

– Unclustered:

- To traverse: $\log_f(1.5N)$
- To scan: $\sim M$ random IO

Suppose we are using a B+ Tree index with:

- Fanout f
- Fill factor $2/3$

Plugging in some numbers

To simplify:

- Random IO = ~10ms
- Sequential IO = free

- Clustered:
 - To traverse: $\text{Log}_F(1.5N)$
 - To scan: 1 random IO + $\left\lceil \frac{M-1}{P} \right\rceil$ sequential IO

~ 1 random IO = 10ms

- Unclustered:
 - To traverse: $\text{Log}_F(1.5N)$
 - To scan: ~ M random IO

~ **M** random IO = M*10ms

- If $M = 1$, then there is no difference!
- If $M = 100,000$ records, then difference is ~10min. Vs. 10ms!

If only we had good estimates of M...

HISTOGRAMS & IO COST ESTIMATION

IO Cost Estimation via Histograms

- For **index selection**:
 - What is the cost of an index lookup?
- Also for **deciding which algorithm to use**:
 - Ex: To execute $R \bowtie S$, which join algorithm should DBMS use?
 - What if we want to compute $\sigma_{A>10}(R) \bowtie \sigma_{B=1}(S)$?
- In general, we will need some way to *estimate intermediate result set sizes*

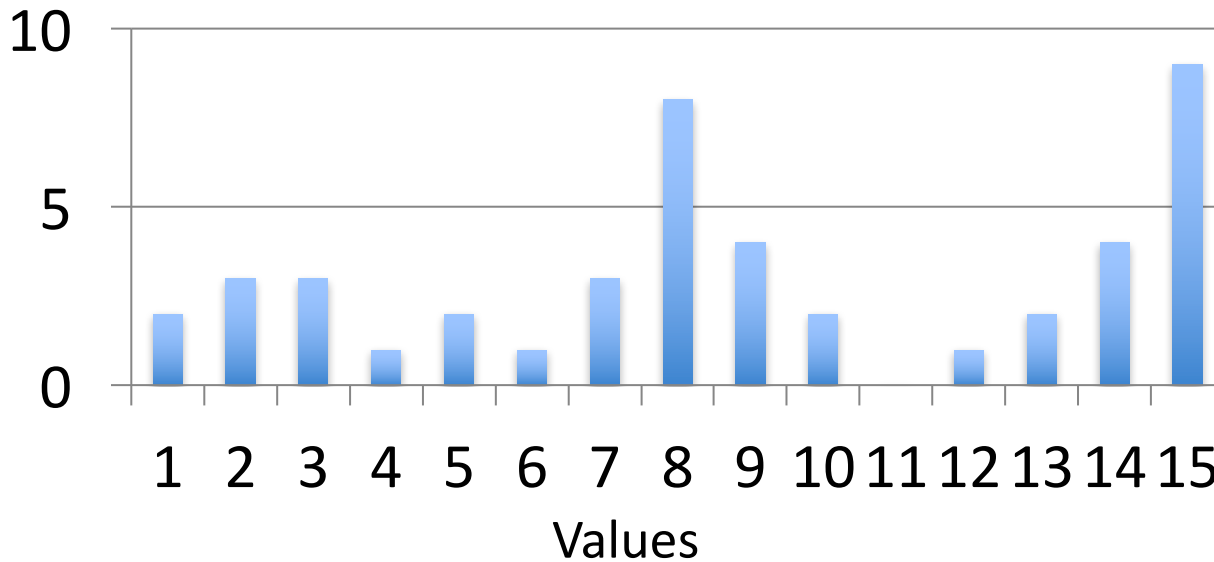
Histograms provide a way to efficiently store estimates of these quantities

Histograms

- A histogram is a set of value ranges (“buckets”) and the frequencies of values in those buckets occurring
- How to choose the buckets?
 - Equiwidth & Equidepth
- Turns out high-frequency values are **very** important

Example

Frequency



How do we
compute how
many values
between 8 and
10?
(Yes, it's obvious)

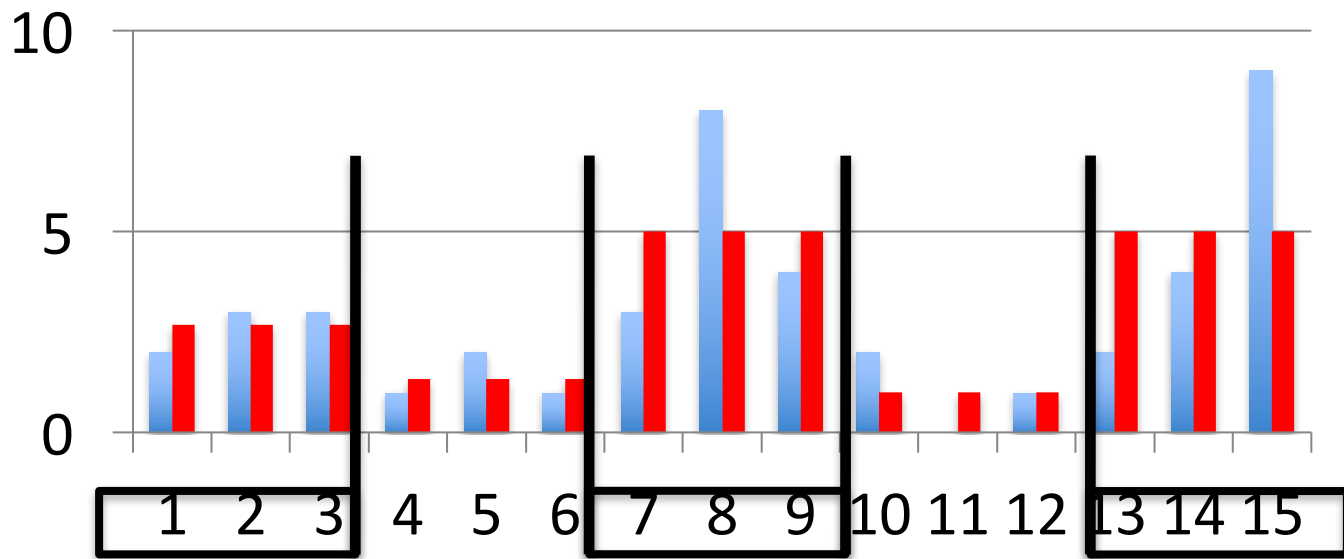
Problem: counts take up too much space!

Fundamental Tradeoffs

- Want high resolution (like the full counts)
- Want low space (like uniform)
- Histograms are a compromise!

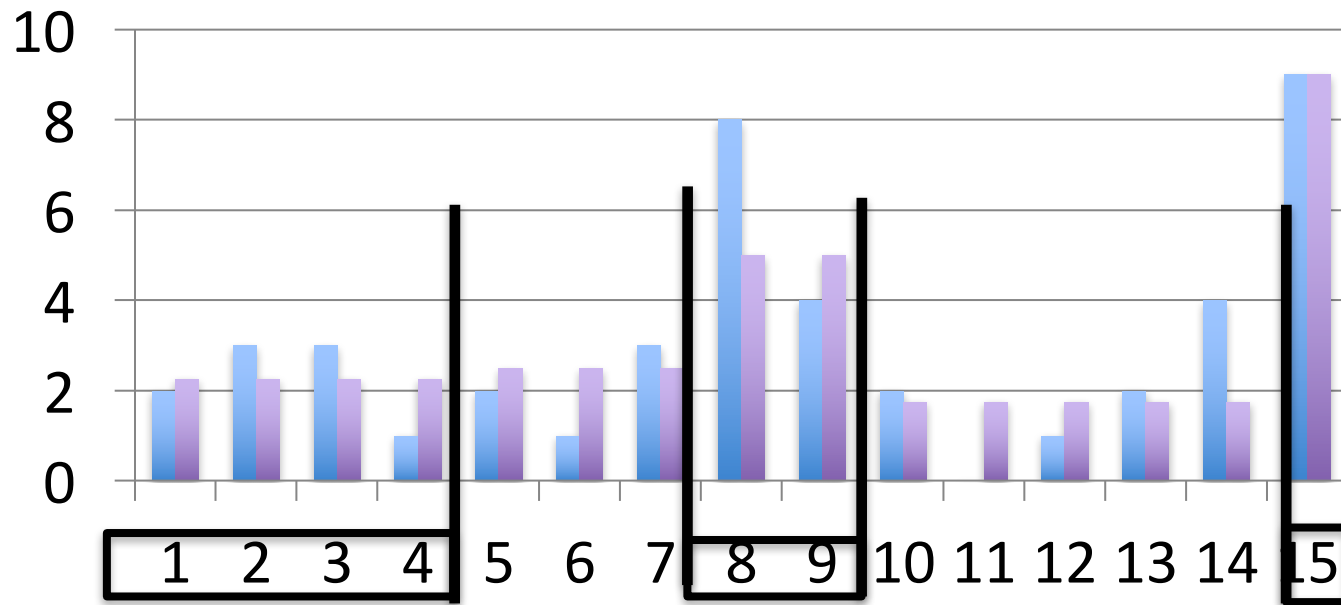
So how do we compute the “bucket” sizes?

Equi-width



All buckets roughly the same width

Equidepth



All buckets contain roughly the same number of items (total frequency)

Histograms

- Simple, intuitive and popular
- Parameters: # of buckets and type
- Can extend to many attributes (multidimensional)

Maintaining Histograms

- Histograms require that we update them!
 - Typically, you must run/schedule a command to update statistics on the database
 - Out of date histograms can be terrible!

Acknowledgement

- Some of the slides in this presentation are taken from the slides provided by the authors.
- Many of these slides are taken from cs145 course offered by Stanford University.