Query Planning: Part 1

Basic Architecture

- ~~Application Layer - what most users see, talks SQL~~
- Parsing/Planning Layers - the intelligence - NEXT
- ~~Runtime or execution Layer - the brawn~~
- ~~Storage Layer - where data resides, may include simple access layer~~

Applications

Parsing

Planning

Processing

Data Access

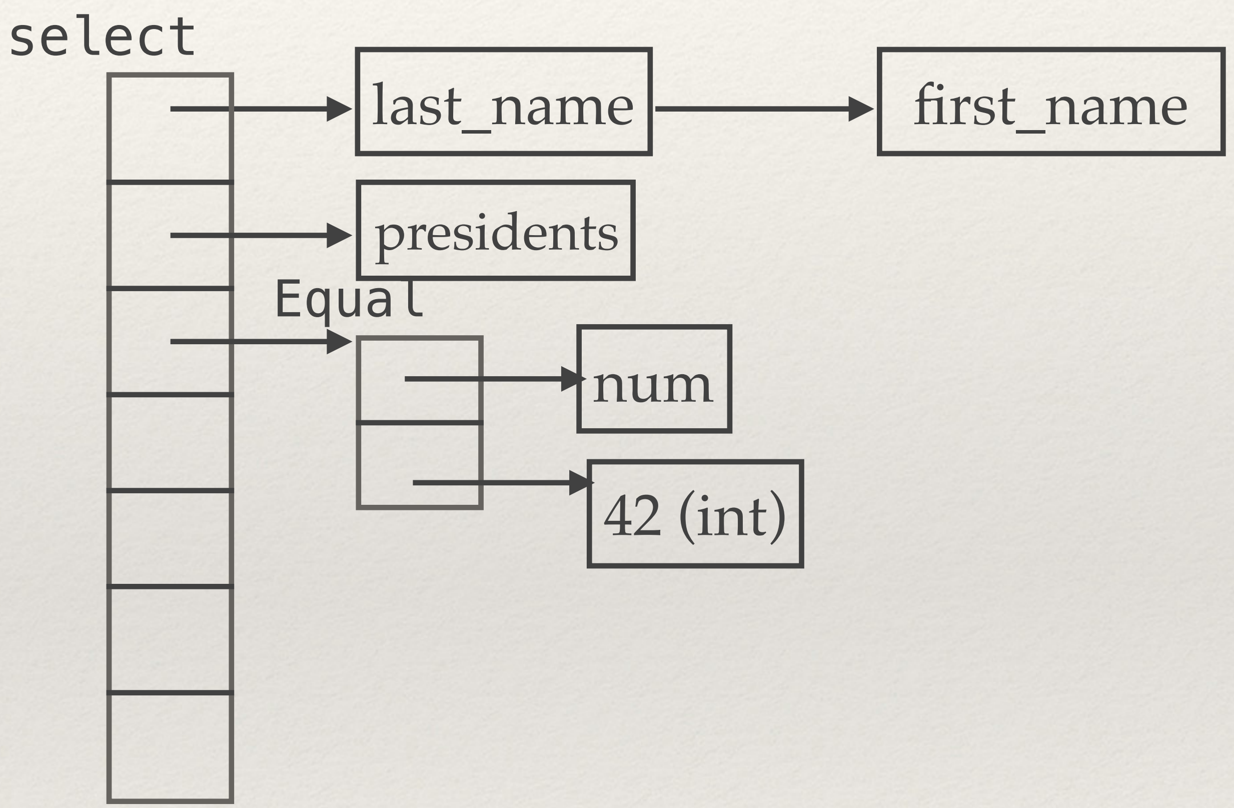
Data in SSD/HDD

Simple Example

- SELECT last_name, first_name
  FROM presidents
  WHERE num = 42;

Step 1: Parsing

```
SELECT last_name, first_name
FROM presidents
WHERE num = 42;
```

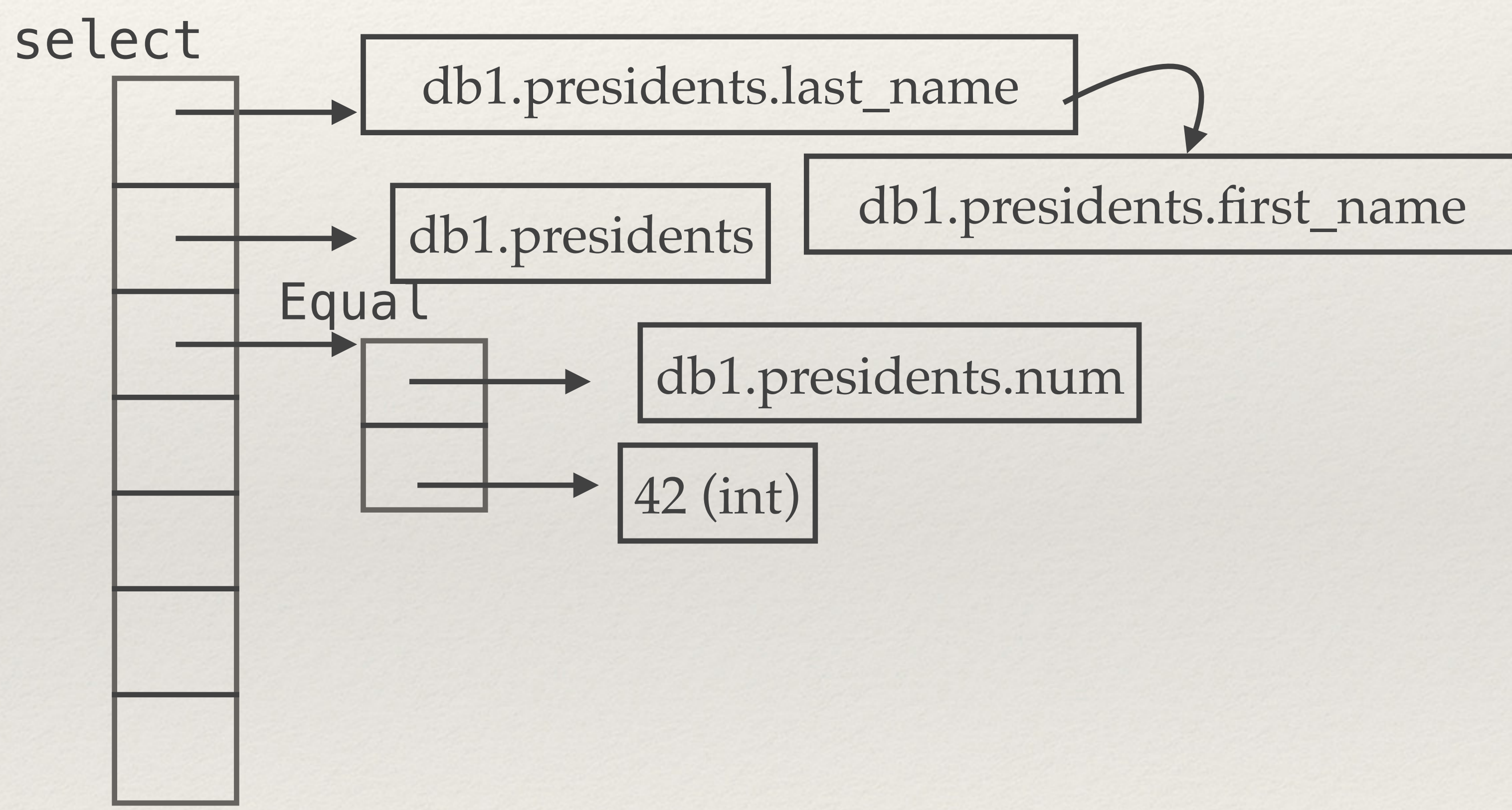


- The SQL is given to a parser that makes a parse tree
- simple syntax errors

Step 2: Resolving the references

- The references are looked up in the catalog a.k.a. (data dictionary)
- errors if things are not found
- E.g.
  - presidents table / view must exist
  - last_name, first_name, num have to be columns from presidents table / view

```
SELECT last_name, first_name
FROM presidents
WHERE num = 42;
```



System Catalog

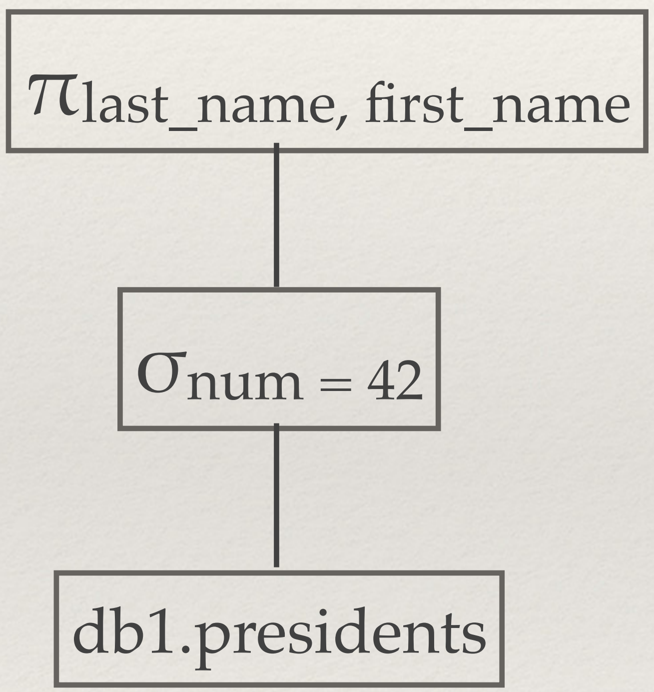
- Set of tables that maintain metadata
  - Users/Databases/Schemas
  - Tables, Views (names, # columns, view_def/pointer to file)
  - Indexes (name, table(s), columns, kind)
  - Attributes (name, type, nullability)
  - Access Rights
  - Statistics
  - User defined functions
  - ….

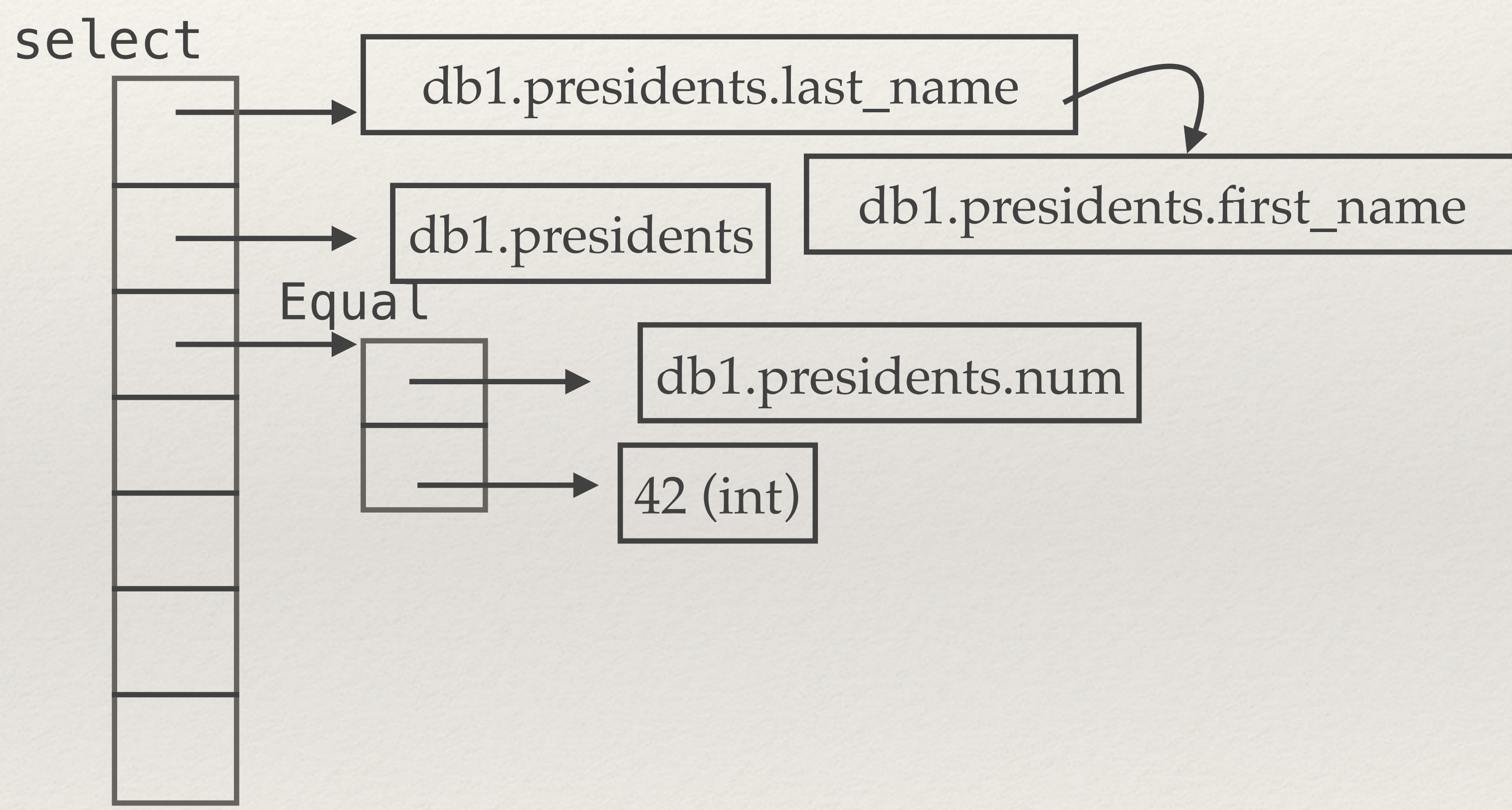
Resolution Process

- The Planner has to issue (highly optimized) queries against the catalog to get all the needed information
- Multiple iterations (e.g. once we get a view, we need to get all the view references resolved)
- At the end: no views etc.
  - all references are to base tables and their columns

SQL-ParseTree-RA Operator Tree

```
SELECT last_name, first_name
FROM presidents
WHERE num = 42;
```





The Optimization Process

- Goal: To generate a query execution plan that is as optimal as practicable
- Both rules-based and cost-based
- Rules-based - where the more optimal option is clear
- Cost-based - where the options have to be compared
- Think of it as transformation of “original RA” tree to an augmented (and hopefully optimal)RA tree

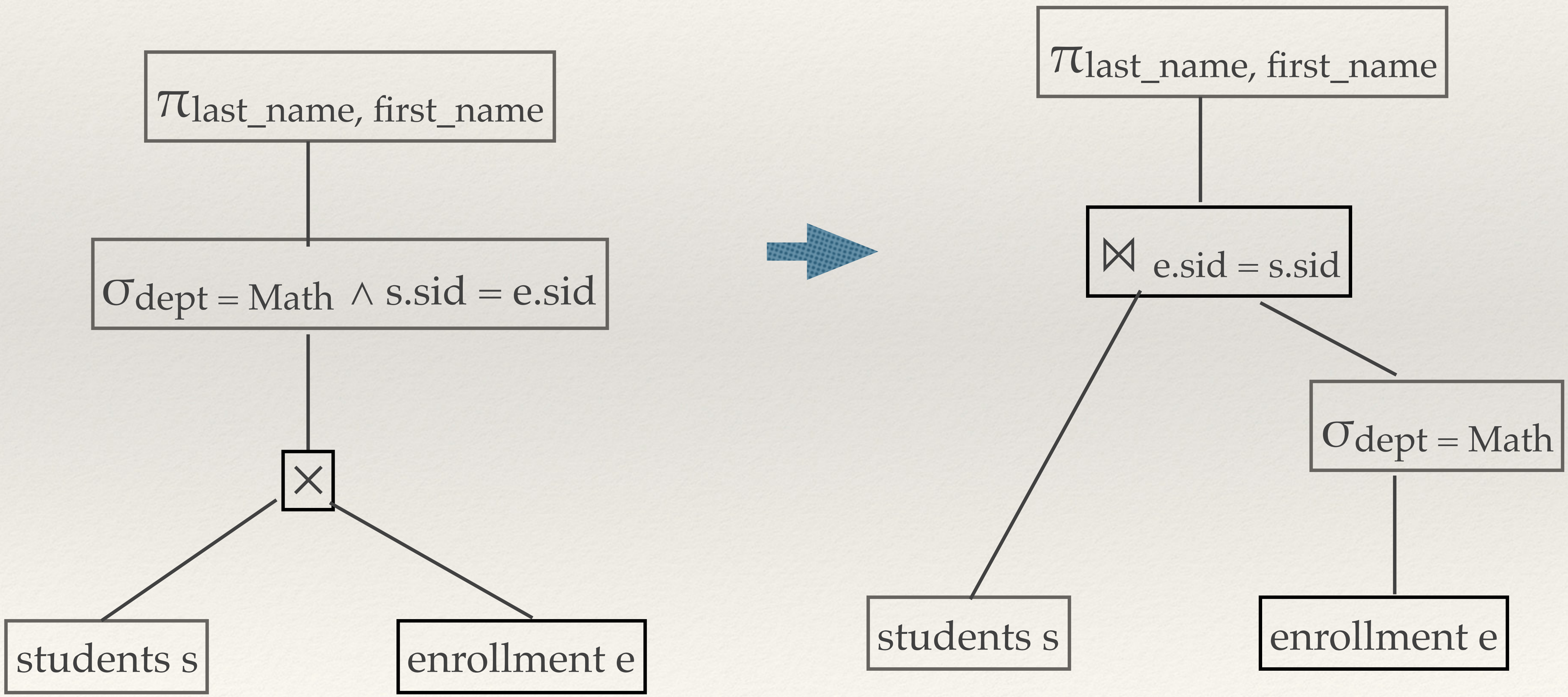
Basic SPJ Query

- This query has the three basic relational algebra operations
- select i.e. condition
- project i.e. choosing a subset of columns and keeping distinct values
- join (equi-join) on FK-PK relationship

```
SELECT DISTINCT last_name, first_name
FROM students s, enrollment e
WHERE s.sid = e.sid
AND e.dept = ‘Math';

 last_name | first_name
-----------+------------
 Bush      | George
```

An SPJ Query



$\pi_{\text{last\_name, first\_name}}$
$\sigma_{\text{dept = Math}} \wedge \text{s.sid = e.sid}$
$\times$
students s
enrollment e
$\pi_{\text{last\_name, first\_name}}$
$\bowtie$ e.sid = s.sid
$\sigma_{\text{dept = Math}}$
students s
enrollment e

Query Optimization

- Goal: to find equivalent relational operator tree that would produce the desired result using least resources and/or most quickly
- If the equivalent transformation is "obvious" then we can create a rule, e.g. "push selections to the table"
- If the equivalent transformations are not obvious we need to compare costs of different ways of doing things

Cost Estimation

- Cost: number of I/O's + CPU + messaging
  - I/O costs dominate
  - All proportional to size of input being processed
  - Row size (easy!) x number of rows (hard)
- Estimate I/O cost for performing operation given the input estimates, but CPU cost are usually estimated too
- Then produce the output size estimate (cardinality + row size) and whether it's sorted (and the sort key)
- Pipelining can affect the I/O cost

Result Size Estimation - Single Table

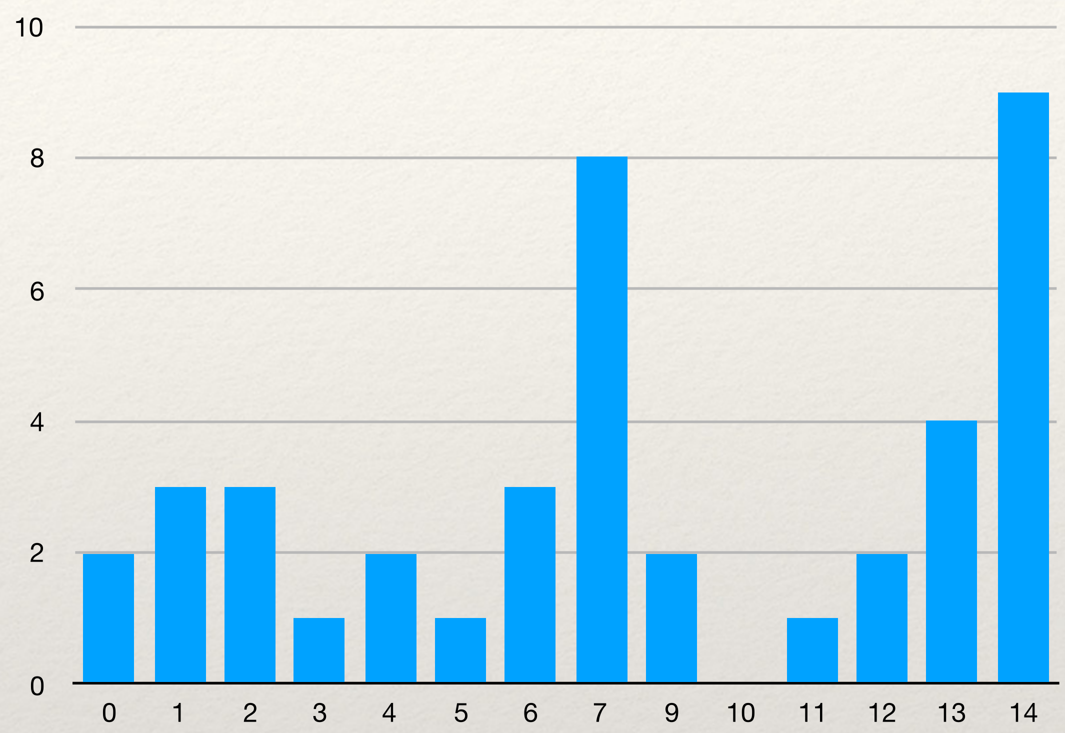
- Est. Result Size = # Rows * Estimated Reduction Factor
- WHERE column = value
  - Estimate: #Rows * (1/#unique values)
  - #unique values estimated from statistics
    - statistics on the column would normally keep number of unique values, min, max and possibly a histogram
    - could sample a page
    - else default to something, say #Rows * (1/10)

Result Size Estimation - contd.

- WHERE column > value
  - Estimate: #Rows * [(max - value) / (max - min)]
  - needs with basic statistics or histogram
- WHERE column IN (v1, v2)
  - same as column = v1 OR column = v2
  - Add up the equality estimates
  - heuristic: limit to #Rows/2

Data Distribution

- Basic Statistics
  - min = 0
  - max = 14
  - #unique = 14
  - #rows = 45

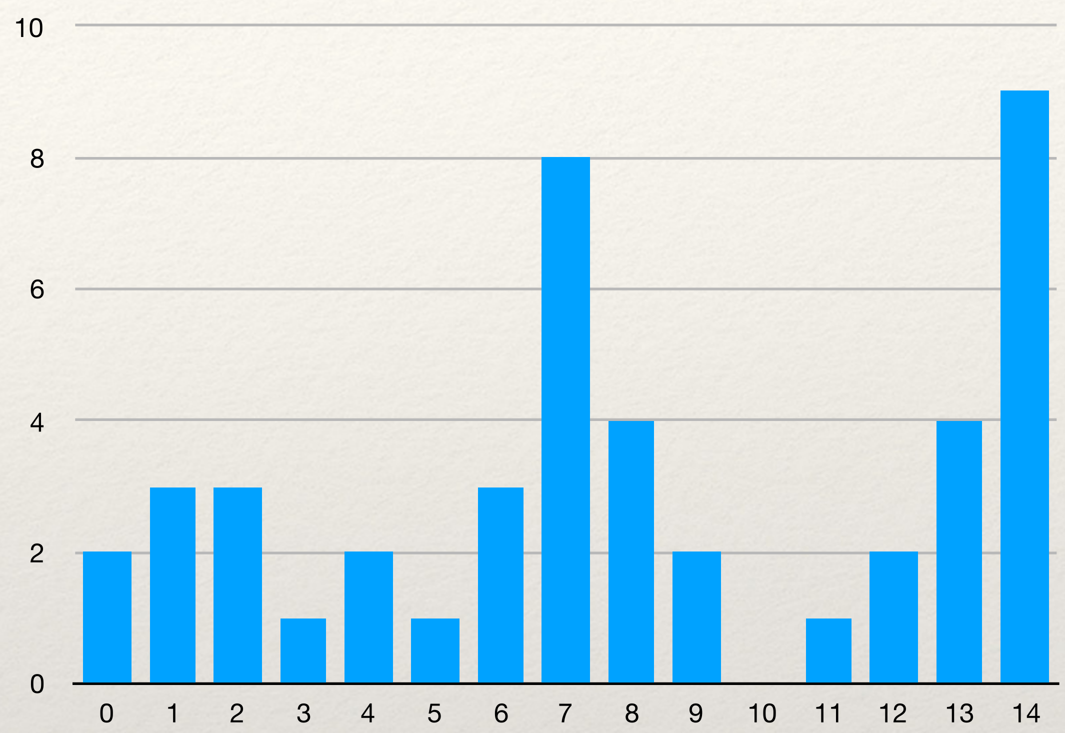


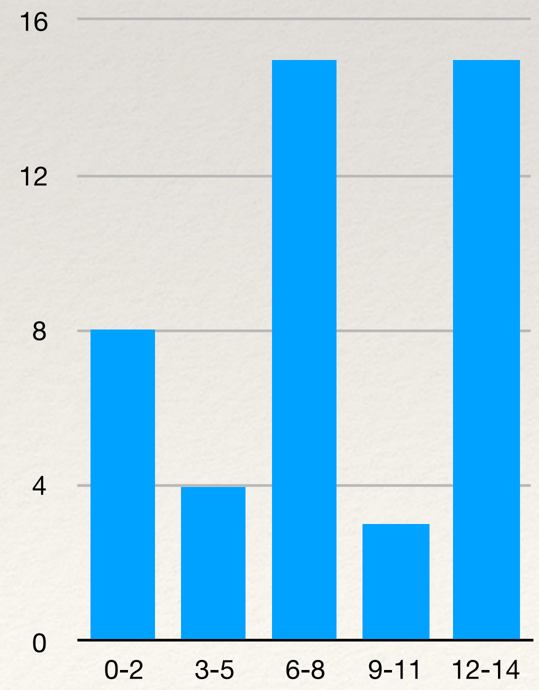
Using Statistics

- Basic Statistics
  - min = 0
  - max = 14
  - #unique = 14
  - #rows = 45
- Estimate # rows
  - value = 7
  - value > 12
  - value in (5,11)

(Equi-width) Histogram

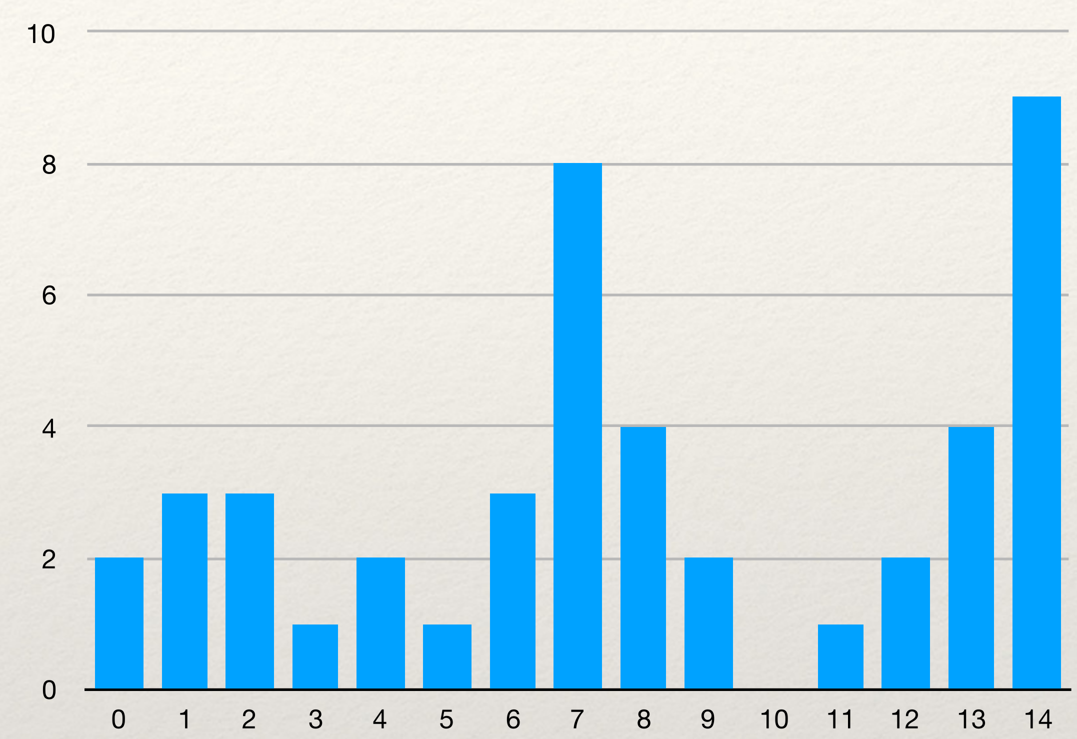
- Basic Statistics
  - min = 0
  - max = 14
  - #unique = 14
  - #rows = 45

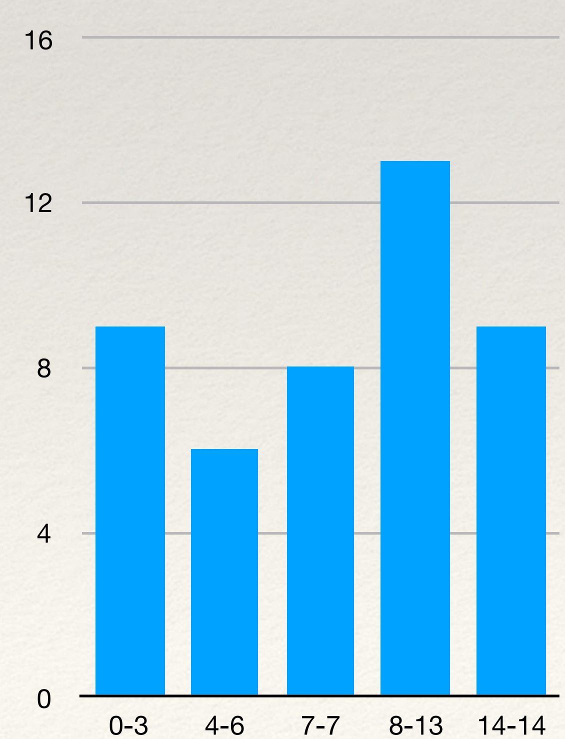




(Equi-height) Histogram

- Basic Statistics
  - min = 0
  - max = 14
  - #unique = 14
  - #rows = 45





Join Size Estimation

- Cross Join of R, S, #rows = #rows(R) * #rows(S)
- WHERE R.column1 = S.column2
  - Assuming PK-FK join
    - i.e. joining columns are a key to one of the tables
    - #rows in result = min(#rows(R), #rows(S))

Join Size Estimation - General

- Assume A is the joining attribute
  - $n_A(R)$ and $n_A(S)$ are number of unique values of A in R and S respectively
- For a row r in R, it would join with #rows(S)/$n_A(S)$
  - => total row estimate is #rows(R)*#rows(S)/$n_A(S)$
  - By symmetry it's also #rows(R)*#rows(S)/$n_A(R)$
- Normally we expect $n_A(S)=n_A(R)$, but if there are non-joining values, one would use the smaller of the two estimates
- Note that this gives the same estimate for FK-PK case
- Non-equijoins are estimated cross joins followed by estimate for the individual conditions

Join Size Estimation and Histograms

- Histograms make the estimates more accurate

$$Estimate = \sum_{bucket=1}^{n} n(R) * n(S) / max(n_A(R), n_A(S))$$

- No Histogram Estimate = 144, Actual Value = 223

Equi-width Histogram Estimate

Bucket	Freq	Estimate
0-2	8	21.3
3-5	4	5.3
6-8	15	75.0
9-11	3	3.0
12-14	15	75.0
	45	179.7

Equi-depth Histogram Estimate

Bucket	Freq	Estimate
0-3	9	20.3
4-6	6	12.0
7-7	8	64.0
8-13	13	28.2
14-14	9	81.0
	45	205.4

Other Estimates

- Projection/Group By estimate
  - Very hard unless there is direct statistics available for the group by/projected columns, sampling is of limited help
- Set operations - fairly approximate
  - UNION: #rows(R) + #rows(S)
  - INTERSECT: min(#rows(R), #rows(S))
  - EXCEPT: #rows(R)
- Outer joins - upper bounds
  - R left outer join S: estimate(R join S) + #rows(R), ROJ is symmetric
  - FULL OJ: estimate(R join S) + #rows(R) + #rows(S)

Histograms

- Besides nRows, min, max, unique count, capture the distribution of data
- For each bucket
  - frequency for range of values
  - number of unique values in bucket
- equi-depth vs. equi-width
- equi-depth usually captures more information by keeping more details about frequent values
- Some databases may even keep separate info for modal value

Transformations

- Two expressions are equivalent if and only if they produce the same result on every legal instance
- The goal is to find the least costly but equivalent expression for evaluating the query
- Most are intuitive

Selections & Projections

- $\sigma_{c1 \wedge c2 \wedge c3}(R) = \sigma_{c1}(\sigma_{c2}(\sigma_{c3}(R)))$
- since $\wedge$ is commutative $\sigma_{c1 \wedge c2}(R) = \sigma_{c2 \wedge c1}(R)$
  - $=> \sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
- reason why we convert conditions to CNF
- A sequence of projections can be reduced to final one
  - $\pi_{L1}(\pi_{L2}(\pi_{L3}(R))) = \pi_{L1}(R)$

Joins and Cross Products

- Cross product and natural (equi-) join are commutative
  $R \times S = S \times R$ and $R \bowtie S = S \bowtie R$
- They are also associative
  $R \times (S \times T) = (R \times S) \times T$ and $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$
- Together, we can join them in any order we choose

Other SPJ Equivalences

- $\sigma_{c1}(\pi_{L1}(R)) = \pi_{L1}(\sigma_{c1}(R))$ if selection only needs attributes retained by projection
- $\sigma_c(R \bowtie S) = \sigma_c(R) \bowtie S$ if c contains attributes from R only
- Generalizing the above - pushing conditions through joins
  - $\sigma_{c1 \wedge c2 \wedge c3} (R \bowtie S) = \sigma_{c1} (\sigma_{c2}(R) \bowtie \sigma_{c2}(S))$
    - where c2 contains attributes only from R
    - c3 contains attributes only from S
    - c1 contains attributes from both

Other SPJ Equivalences

- Pushing Projections
  - $\pi_L(R \bowtie_c S) = \pi_{L1}(R) \bowtie_c \pi_{L2}(S)$ where
    - L1 is subset of attributes of R that are in L
    - L2 is subset of attributes of S that are in L
    - All attributes in C must be there in L
- Generalizes to
  - $\pi_L(R \bowtie_c S) = \pi_L(\pi_{L1}(R) \bowtie_c \pi_{L2}(S))$ where
    - L1 is subset of attributes of R that are in L or C
    - L2 is subset of attributes in S that are in L or C

Basic Optimization Strategy

- Enumerate alternative plans
- Pick the best (least estimated cost)
- Equivalences are the basis for alternate plans. This is combined with alternative operator algorithms and alternate ways to access a single table

Single Table Access

- Enumerate all ways to evaluate a single table access
  - Scan and evaluate condition
  - Use the primary (clustered) index if applicable
  - Use one secondary index if applicable
  - Use multiple secondary indices if applicable
  - Use other structures (e.g. materialized views) if applicable

Joins

- left - right asymmetry in hash and nested loop
- Assume hash and sort-merge
- 2 table joins => $R \bowtie S$ (hash), $S \bowtie R$ (hash), $R \bowtie S$ (SM)
- 3 tables - just one algorithm, ignore asymmetry
  - $R \bowtie (S \bowtie T)$, $(R \bowtie S) \bowtie T$, $(R \times T) \bowtie S$