Goals for Today

- Continue to reason about estimating the result cardinality for selections and joins
  - System R heuristics
  - More advanced: histograms – Lab 3!
- Begin to explore the search space explosion for alternate query plans
- Discuss the influence of index selection

Cost-based Query Sub-System

Histograms: Finer-Grained Statistics

- For better RF estimation, many systems use histograms
- Histogram is *approximation of a data distribution*
- **Example:** ratings of Sailors (40,000 total tuples)

---

CS 133: Databases

Fall 2016
Lec 14 – 10/24
Prof. Beth Trushkowsky
Equi-width vs. Equi-depth Histograms

- **Equi-width**
  - # values represented by each bucket is the same

- **Equi-depth**
  - # of records in each bucket is ~same

Exercise 2: Histograms

Diagram from Lab 3

Be careful with:
- Bucket size and off-by-one issues
- Selectivity of values outside the min/max

Equivalent Relational Algebra Expressions

- Can write the same query multiple ways!
  - These alternate versions are akin to different possible logical query plans

- Good rules of thumb:
  - “Push” down selections
  - Avoid cross-products

What are the legal transformations for an expression?

Query Optimizer algorithm

- Goal: given a a query, the optimizer wants to
  - Enumerate query plans to consider
  - Compare plans and choose the “best” one

- How about this algorithm?
  - Step 1: enumerate the space of all possible plans
  - Step 2: estimate cost for each plan
  - Step 3: choose the plan with lowest cost

Why?
Relational Algebra Equivalences

Selections:
\[ \sigma_{c_1 \land \ldots \land c_n}(R) \equiv \sigma_{c_1}(\ldots \sigma_{c_n}(R)) \quad \text{(Cascade)} \]
\[ \sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R)) \quad \text{(Commutate)} \]

Projections:
\[ \pi_{a_1}(R) \equiv \pi_{a_1}(\ldots \pi_{a_n}(R)) \quad \text{(Cascade)} \]
(if \( a_n \) includes \( a_{n-1} \) includes… \( a_1 \))

A projection could commute with a selection, e.g.,
\[ \pi_{a_1}(\sigma_{c}(R)) \equiv \sigma_{c}(\pi_{a_1}(R)) \quad \text{... if condition } c \text{ acts only on attributes in } a \]

R.A. Equivalences: Joins

Joins:
\[ (R \Join S) \equiv (S \Join R) \quad \text{(Commutative)} \]
\[ R \Join (S \Join T) \equiv (R \Join S) \Join T \quad \text{(Associative)} \]

Selection between attributes of the two arguments of a cross-product converts cross-product to a join:
\[ \sigma_{R.a=S.b}(R \times S) \equiv (R \Join_{R.a=S.b} S) \]

Exercise 3-4

Selection Push: selection on attributes of \( R \) commutes with \( R \Join S \):
\[ \sigma(R \Join S) \equiv \sigma(R) \Join S \]

Projection Push: A projection applied to join of \( R \) and \( S \) can be pushed before the join by:
- retaining only attributes of \( R \) and \( S \) needed for the join,
- or are kept by the projection
\[ \pi_{R.a,S.b}(R \Join_{R.a=S.b} S) \equiv (\pi_{R.a}(R)) \Join_{R.a=S.b} \pi_{S.b}(S) \]

- Convert cross-product to join with \( R.a=S.c \)
- Commute the select condition \( R.a > 2 \) with join
- Note: cannot push projection \( R.a \) before join
  - But could cascade the projection: project \( R.a,c \) before join, then project \( R.c \) after select

- Joining Boats and Sailors first would yield a lot of tuples, since this would become a cross-product!
Enumeration of Alternative Plans

- There are two main cases:
  - Single-relation plans (unary operators only)
  - Multiple-relation plans

- For unary operators:
  - For a scan, each available access path (sequential scan / index) is considered; one with the least estimated cost is chosen
  - Consecutive Scan, Select, Project and Aggregate operations can be typically pipelined

Exercise 5: Plan Enumeration (Single Relation)

- Take advantage of the index on <age,rating> for an index-only query plan!

- Already sorted on age, so can pipeline into Aggregate operator to get average rating per age group

Physical DB Design

- Query optimizer does what it can to use indexes, clustering, and operator implementations

- Database Administrator (DBA) is expected to set up physical design well
  - Consider which indexes to create

Index Selection

- A greedy approach:
  - Consider most important queries in turn.
  - Consider best plan using the current indexes
  - See if better plan is possible with an additional index.
  - If so, create it.

- But consider impact on updates!
  - Indexes can make queries go faster, updates slower.
  - Require disk space, too.

Good DBAs understand query optimizers very well!
Example: choosing indexes

```
SELECT E.ename, D.mgr
FROM   Emp E, Dept D
WHERE  E.dno=D.dno AND D.dname='Toy'
```

- Hash index on *D.dname* supports ‘Toy’ selection
  - Good access path as outer relation
  - Given this, index on *D.dno* is not needed
- Hash index on *E.dno* allows us to get matching (as inner) Emp tuples for each selected (outer) Dept tuple.

- What if WHERE clause included **AND E.age=25**?
  - Could retrieve Emp tuples using index on *E.age*, then join with Dept tuples satisfying *dname* selection.
  - So, if *E.age* index is already created, this query provides much less motivation for adding an *E.dno* index.

Clustering vs. Not Clustering

```
SELECT E.ename, D.mgr
FROM   Emp E, Dept D
WHERE  E.dno=D.dno AND D.dname='Toy'
```

- If *dname* highly selective, okay if unclustered
- Makes sense for index on *E.dno* to be clustered if not primary key of Emp
  - Impact of unclustered-ness amplified for *inner* relation

Clustering vs. Not Clustering

```
SELECT E.ename, D.mgr
FROM   Emp E, Dept D
WHERE  E.dno=D.dno AND D.dname='Toy'
```

- Suppose many tuples have *E.age* > 10
- Clustered indexes choice?
  - *E.age*
  - *E.dno*

Enumerating Multi-Relation Plans

```
SELECT E.dno, COUNT (*)
FROM   Emp E
WHERE  E.age > 10
GROUP BY E.dno
```

- Suppose we have *N* relations
  - Let’s ignore the space of different join algorithms for a moment
  - Recall: associative and commutative rules mean we can apply joins in any order

- How many join orders? Example: *N=3, {A,B,C}*
  - How many tree shapes?
  - Given a tree shape, how many leaf orderings?

For both tree shapes, can have 6 orderings of relations in the leaves
Exercise 6: Join Orders

- Leaf orderings given a shape? N!

- Tree shapes, for a fixed ordering of 4 relations
  - 1 left-deep and linear
  - 1 right-deep and linear
  - 1 bushy
  - 2 linear

Number of Join Orders

- Leaf permutations: n!
- Tree shapes: Catalan numbers
  \[ C(n) = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} \]
- Join orders(n) = n! * C(n-1)


Lab 3: SimpleDb Optimizer

Exercise 1: describe formation of physical query plan

Exercise 2-3: histogram for statistics and setting up TableStats

Exercise 4-5: join statistics and dynamic programming algorithm for join ordering

Statistics, like the Catalog, are memory-only in SimpleDb

Generated when Parser initialized