Query Optimization Steps

John A. Miller

University of Georgia

April 13, 2024

1 Setup

1.1 Schema

```
branch   = Table ("branch", "bname, assets, bcity", "S, D, S", "bname")
customer = Table ("customer", "cname, street, ccity", "S, S, S", "cname")
deposit  = Table ("deposit", "bname, accno, cname, balance", "S, I, S, D", "accno")
loan     = Table ("loan", "bname, loanno, cname, amount", "S, I, S, D", "loanno")
```

1.2 Statistics

The table sizes (number of tuples per table) are the following:

$$\begin{aligned} n_b &= 100 \\ n_c &= 1000 \\ n_d &= 2000 \\ n_l &= 3000 \end{aligned}$$

The number of distinct values for attribute A in relation r is given by $V(A, r)$.

$$\begin{aligned} V(ccity, customer) &= 10 \\ V(bcity, branch) &= 10 \end{aligned}$$

2 Query

List the names of customers with a deposit account at a branch located in Athens.

SQL Query:

```
SELECT d.cname
FROM deposit d, branch b
WHERE b.bcity = 'Athens' and d.bname = b.bname
```

Translation to Relational Algebra:

$$\pi_{cname}(\sigma_{bcity='Athens' \text{ and } d.bname=b.bname}(deposit \times branch))$$

3 Optimization Steps

1. **Direct Translation from SQL**.

   SELECT $\rightarrow \pi$, FROM $\rightarrow \times$, WHERE $\rightarrow \sigma$.

   π_{cname}
   | | | **o**: $\boxed{\frac{n_\times}{V(bcity, branch)n_b}} = \frac{200,000}{10 \cdot 100} = 200$
   $\sigma_{bcity='Athens' \text{ and } d.bname=b.bname}$
   | **i**: $\boxed{n_\times} = 200,000$
   | | **o**: $\boxed{n_\times = n_b n_d} = 100 \cdot 2000 = 200,000$
   $\times$
   | \ **i**: $\boxed{n_b(1+n_d)} = 100(1+2000) = 200,100$
   | \
   | \
   deposit *branch*
   (2000) (100)

   TOTAL = 600,300 tuple accesses
   SPEED-UP = 1

2. **Convert Cartesian Products to Joins**.

   Factor predicate out of conjoined predicates in select and apply with product ($\times$) to form join ($\bowtie$).

   π_{cname}
   | | | **o**: $\boxed{\frac{n_\bowtie}{V(bcity, branch)}} = \frac{2000}{10} = 200$
   $\sigma_{bcity='Athens'}$
   | **i**: $\boxed{n_\bowtie} = 2000$
   | | **o**: $\boxed{n_\bowtie = n_d} = 2000$
   $\bowtie_{d.bname=b.bname}$
   | \ **i**: $\boxed{n_b(1+n_d)} = 100(1+2000) = 200,100$
   | \
   | \
   deposit *branch*
   (2000) (100)

   TOTAL = 204,300 tuple accesses
   SPEED-UP = 2.94

3. **Move Selects Down the Query Evaluation Tree**.

   Filter tuples as early as possible by moving selects (σ) as far down the tree as possible.

   π_{cname}
   |
   |
   | **o**: $\boxed{n_{\bowtie} = n_d p_{match}} \;=\; 2000 \cdot 0.1 \;=\; 200$
   $\bowtie_{d.bname=b.bname}$
   | \ **i**: $\boxed{n_\sigma(1 + n_d)} \;=\; 10(1 + 2000) = 20,010$
   | \
   | \ **o**: $\boxed{n_\sigma = \dfrac{n_b}{V(bcity, branch)}} \;=\; \dfrac{100}{10} \;=\; 10$
   | $\sigma_{bcity='Athens'}$
   | | **i**: $\boxed{n_b} = 100$
   | |
   | |
   deposit *branch*
   (2000) (100)

   TOTAL = 20,320 tuple accesses
   SPEED-UP = 29.54

4. **Replace Sequential Select with Indexed Select**.

   n_{idx} = number of nodes accessed in unique index (see formulas at end)

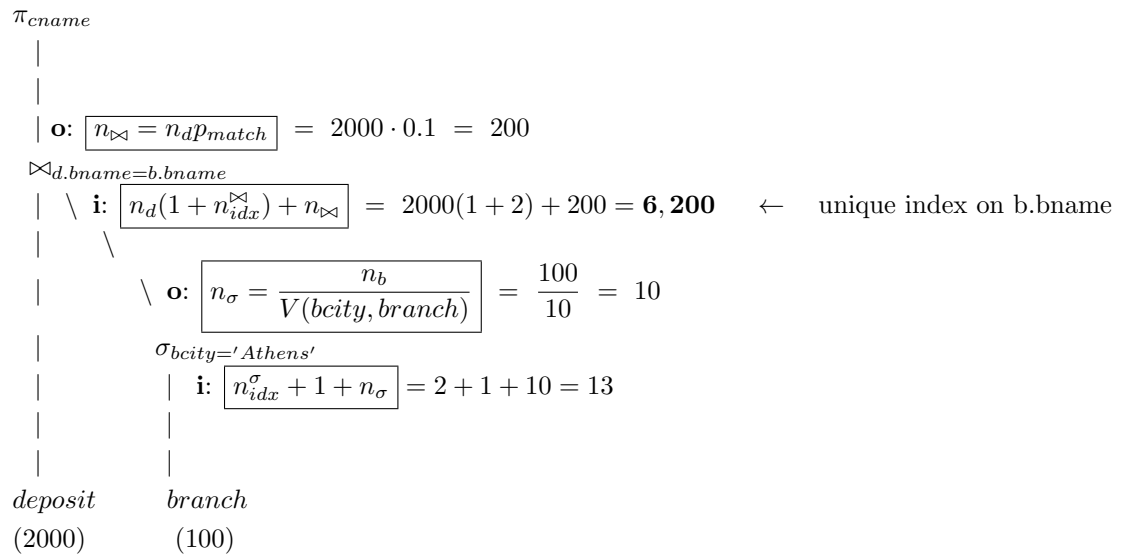
   Use $n_{idx} + 1$ for non-unique index (accounts for adapter node holding multiple tuple references)

   π_{cname}
   |
   |
   | **o**: $\boxed{n_{\bowtie} = n_d p_{match}} \;=\; 2000 \cdot 0.1 \;=\; 200$
   $\bowtie_{d.bname=b.bname}$
   | \ **i**: $\boxed{n_\sigma(1 + n_d)} \;=\; 10(1 + 2000) = 20,010$
   | \
   | \ **o**: $\boxed{n_\sigma = \dfrac{n_b}{V(bcity, branch)}} \;=\; \dfrac{100}{10} \;=\; 10$
   | $\sigma_{bcity='Athens'}$
   | | **i**: $\boxed{n^\sigma_{idx} + 1 + n_\sigma} = 2 + 1 + 10 = \mathbf{13}$ $\leftarrow$ non-unique index on bcity
   | |
   | |
   deposit *branch*
   (2000) (100)

   TOTAL = 20,233 tuple accesses
   SPEED-UP = 29.67

5. **Replace Nested Loop Join with Indexed Join using a Unique Index**.

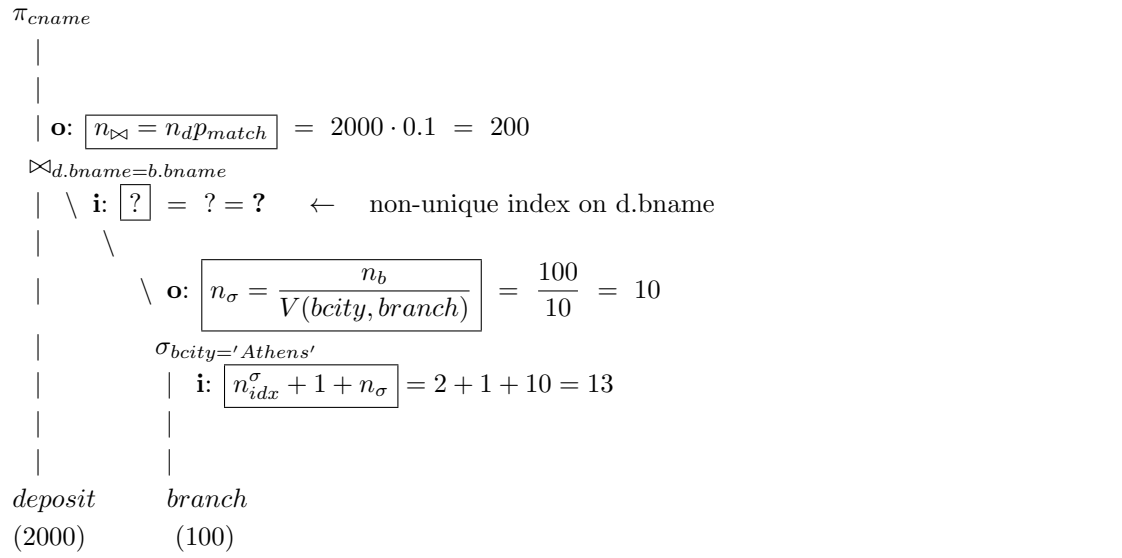
   Typical Case: Use the unique index on the primary key to replace inner loop in Nested Loop Join with index look-up.



   TOTAL = 6,423 tuple accesses
   SPEED-UP = 93.46

6. **Replace Nested Loop Join with Indexed Join using a Non-Unique Index**.

   Alternative: Use the non-unique index on the foreign key to replace inner loop in Nested Loop Join with index look-up.



   TOTAL = ? tuple accesses
   SPEED-UP = ?

4 Estimating Accesses

4.1 Estimating Tuple Accesses - No Index Case

4.1.1 Estimating Tuple Accesses for Select Operations

Without an index and ignoring the unique key case that allows for early termination of the search, a select operation will require a full table scan, so the input/processing (**i:**) equals the number of tuples in relation r, i.e., n_r. The number of tuples output (**o:**) depends of the selectivity of the predicate $P, Sel(P)$..

$$\sigma_P(r) \text{ has } \mathbf{i} : \boxed{n_r} \text{ and } \mathbf{o} : \boxed{n_r Sel(P)}$$

Common Case: Predicate P is $A = a$ and $B = b$, then the output (**o:**) is given by

$$\mathbf{o} : n_\sigma = \frac{n_r}{V(A,r)V(B,r)}$$

Assumptions: Attributes A and B are independent and their values are uniformly distributed.

4.1.2 Estimating Tuple Accesses for Join Operations

One may view a join operation as repeated select operations.

$$\begin{array}{l} q = \{\} \\ \textbf{for } t \in r \textbf{ do } q = q + t \text{ cat } \sigma_{A=t[B]}(s) \\ q \end{array}$$

Therefore,

$$r \bowtie_P s \text{ has } \mathbf{i} : \boxed{n_r(1 + n_s)} \text{ and } \mathbf{o} : \boxed{n_r n_s Sel(P)}$$

The n_r is the multiplier due to the for loop, while inside the loop, the 1 is the access to the t tuple in r and the n_s are the accesses to the tuples in s. One may minimize the count by using the smaller relation for the outer loop of the Nested Loop Join.

4.2 Estimating Index Node Accesses

Account for a node access like a tuple access and let TOTAL be the sum of tuple and node accesses.

$$\begin{array}{rcl} n_{idx} & = & 1 + \frac{\alpha}{2} \qquad \text{for Linear Hashing (LH)} \\ & = & h_{min}(V(A,r)) + 1 \qquad \text{for B}^+\text{Trees (BP)} \end{array}$$

where the expected height of the tree (edge count) is typically close to $h_{min}(n)$.

$$h_{min}(n) = \left\lceil \log_p \frac{n}{p-1} \right\rceil$$

The one is added to get the node count, traversing from the root to a leaf.

4.3 Estimating Tuple and Node Accesses - Index Case

4.3.1 Estimating Tuple Accesses for Select Operations

The only change needed when switching to an Indexed Select is add to n_σ the number of node accesses required to find these desired tuples.

$$\sigma_P(r) \text{ has } \mathbf{i}: \boxed{n_{idx} + n_\sigma} \quad \text{unique case}$$
$$\mathbf{i}: \boxed{n_{idx} + 1 + n_\sigma} \quad \text{non} - \text{unique case}$$

4.3.2 Estimating Tuple Accesses for Join Operations

Switching to an Indexed Join involves more complicated formulas and a choice of whether to use a unique index on the primary key or a non-unique index on the foreign key. In any case, the inner loop of the Nested Loop Join is replaced with an index lookup.

$q = \{\}$
for $t \in r$ **do** $q = q + t \text{ cat } index(K_s = t[B], s)$
q

4.3.3 Unique Index on the Primary Key

Consider the case where relation r is the foreign key table and s is the primary key table and therefore its unique index will be used. Starting with the formula for the non-indexed case,

$$r \bowtie_P s \text{ has } \mathbf{i}: \boxed{n_r(1 + n_s)}$$

we still need the outer loop to access the tuples in r and apply the index into relation s. Ideal use of the index, could mean the desired tuples are only accessed once, leading to a total of $n_{\bowtie}$ tuple accesses to relation s ($n_{\bowtie}$ is the number tuples output).

$$r \bowtie_P s \text{ has } \mathbf{i}: \boxed{n_r(1 + n_{idx}) + n_{\bowtie}}$$

4.3.4 Non-Unique Index on the Foreign Key

Consider the opposite case where relation r is the primary key table and s is the foreign key table and its non-unique index will be used.

$$r \bowtie_P s \text{ has } \mathbf{i}: \boxed{?}$$

5 Homework

1. Recompute the TOTAL ACCESSES for Steps 4 and 5 assuming the LH load factor $\alpha = 1$ (was assumed to be 2).
2. Recompute the TOTAL ACCESSES for Steps 4 and 5 assuming the BP order $p = 10$.

3. Determine the **i:** formula for section 4.3.4.

4. Use the above formula to complete Step 6.