

Query Optimization

1

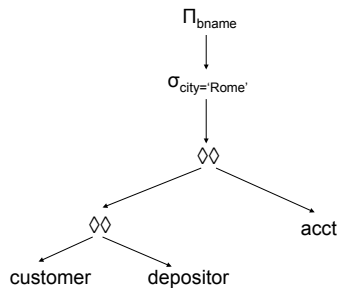
Query Optimization

- Query as **expression** over relational algebraic operations
- Get evaluation (parse) tree
 - Leaves: base relations
 - Interior nodes: operations

2

Example

$\Pi_{\text{bname}} (\sigma_{\text{city}='Rome'} ((\text{customer} \bowtie \text{depositor}) \bowtie \text{acct}))$

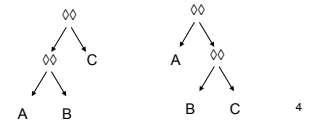


3

Optimization considerations

- Choice of algorithm at each interior node
 - **Cost Estimates**
 - We've just studied analysis
- Rearrange tree
 - Use **algebra** of operations
 - e.g. associativity of JOIN

$$(A \bowtie B) \bowtie C = A \bowtie (B \bowtie C)$$

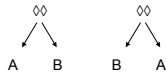


4

Interaction of algorithm choice and tree arrangement

- Convention: for any nested loop join, **left branch** represents **outer relation**
 - Control with commutativity of JOIN

$$(A \bowtie B) = (B \bowtie A)$$



- **Result of an interior node is input to parent**
 - Algorithm affects **properties** of **presentation** of result
 - Sorted?
- **Cost** analysis must proceed **bottom up**

5

Issues

- Need **size estimates** of **result relation**
 - # records per page (**size of record**)
 - # of pages (**# of records**)
 - Note:
 - page size fixed system parameter
 - Duplicates significantly affect # of records
- Need plan for **buffer use**
 - **Materialize result**: write result of interior node **to disk**
 - Costs of writes for intermediate results count!
 - **Intermediate result fits in buffer**
 - Algorithm for parent use this?
 - Can save cost of writing result by child & reading result by parent
 - **Pipeline** result of child as input to parent

6

Pipelining

- Parent and child **execute concurrently**
- Parent and child **share buffer space**
 - k-page shared (sub)buffer
 - child **produces k pages** of output – **Fill buffer**
 - parent **consumes k pages** of input from child – **Empty buffer**
 - **NO disk write cost** child;
 - **NO disk read cost** parent
- Algorithms of child and parent must support this
 - Child: usually does; produce 1 page output at a time
 - Parent: **choice of algorithm critical!**

7

Algorithms for parent - JOIN

- block nested loop?
- Index nested loop?
- Sort-merge
- Hash

8

Algorithms for parent - JOIN

- Block nested loop?
 - Outer relation – ok
 - Read relation once, “chunk” by “chunk”
 - Use shared buffer for “chunk”
 - Inner relation – NO
 - Must re-read *entire* inner relation for every “chunk” of outer
- Index nested loop?
- Sort-merge
- Hash

9

Algorithms for parent - JOIN

- Block nested loop?
 - Outer relation – **OK**
 - Inner relation – **NO**
- Index nested loop?
 - Outer relation – ok – same as Block nested loop
 - Inner relation – NO
 - Using index
- Sort-merge
- Hash

10

Algorithms for parent - JOIN

- Block nested loop?
 - Outer relation – **OK**
 - Inner relation – **NO**
- Index nested loop?
 - Outer relation – **OK**
 - Inner relation – **NO**
- Sort-merge
 - To sort input relation:
 - Can pipeline from child to group of buffer pages for Stage 1 (Stage 1: sorting individual groups to make runs)
 - If child produced in sorted order, pipeline merge
 - Child must be outer relation if duplicates
- Hash

11

Algorithms for parent - JOIN

- Block nested loop?
 - Outer relation – **OK**
 - Inner relation – **NO**
- Index nested loop?
 - Outer relation – **OK**
 - Inner relation – **NO**
- Sort-merge – **OK**
- Hash
 - To partition input relation:
 - Can pipeline from child to buckets in buffer for Stage 1

– **OK**

12

Allocating buffer pages

- If have simultaneous pipelining up tree
 - How many buffer pages for each child-to-parent exchange?
 - Affects speed of algorithms
- Limit number of simultaneous pipelines
- If no pipeline between child and parent **materialize** result of child
 - Child **writes result to disk**
 - Parent **reads from disk**

13

Multi-operation query

- Want **plan**
 - Parse tree
 - Pipelining plan for each edge
 - Algorithm for each interior node (operation)
- To build plan
 - Consider alternatives
 - ALL?
 - Estimate costs
 - Choose “best”
 - Really “good enough”

14

Catalog

- Need info about base relations
- In **catalog**:
 - For each base relation:
 - # tuples
 - # pages
 - List of existing indexes
 - For each index
 - # distinct search-key values
 - # pages
 - For each tree index
 - Tree height
 - high/low search keys

15

Calculating size estimates of result

- Assume
 - independence of attributes of a tuple
 - Uniform distribution of values of each attribute among tuples
- Calculate **reduction factor (RF)** for # tuples of result
 - Examples:
 - $\sigma_{f = \text{constant}}$ and index on attribute f:

$$RF = 1 / (\# \text{ search key values})$$
 - $\sigma_{f > \text{constant}}$ and tree index on attribute f:

$$RF = \frac{(\text{high key value}) - \text{constant}}{(\text{high key value}) - (\text{low key value})}$$
 - Estimate # tuples output as $RF * (\# \text{ tuples input relation})$

16

Reduction factor of joins

- Estimate # tuples of $(R \bowtie S)$ on shared attribute f as
 - $RF * (\# \text{ tuples } R) * (\# \text{ tuples } S)$
 - Looking at join as selection on $R \bowtie S$
- Example: \bowtie for join attribute f
 - If indexes on R.f and S.f
 $RF = 1 / \max(\# \text{ key values } R.f, \# \text{ key values } S.f)$
 - If no indexes, *could* use # distinct values
 - What if real-valued?

17

Size of tuples of result

- If attributes of fixed length, calculate
 - Projection: sizes of attributes retained
 - Cross-product $R \times S$: sum of sizes of tuples in R and S
 - Join with single occurrence equal attributes
 - Projection of Cross-product
 - Selection & Union-compatible set operations: no change
- If attributes of variable length, estimate

18

Planning

- Know how **estimate costs** of algorithms
 - Know how **estimate sizes** of results
 - How use to **make plan** for query eval?
- interact
- determine operation order for expression
 - algebraic equivalences
 - select algorithm for each operation
 - best depends on operation order
- Can't try all possibilities - exponential time

19

Heuristics

Consider k joins: $R_1 \bowtie R_2 \bowtie \dots \bowtie R_k$

- Too many parse trees
 - associativity and commutativity
- Example heuristic: consider only “**Left-deep join trees**”
 - IBM system R 1979
 - determines tree shape, not order R_i
 - why this shape?
 - still a lot of trees: $k!$

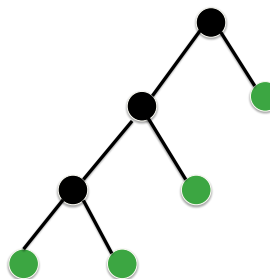
20

Algorithm design

- Observe for $(R_1 \bowtie R_2 \bowtie \dots \bowtie R_{k-1}) \bowtie R_k$:
 - once decide least-cost way do ()
 - actual **order compute w/in () not affect**
 - best choice for () $\bowtie R_k$
 - whether () result **sorted or hashed does** affect best choice for () $\bowtie R_k$
- ⇒ **dynamic programming algorithm**
- walk up left-deep tree

21

Left-deep tree



22

Using dynamic programming

For node distance d from leftmost leaf,

- **estimate lowest cost** of evaluating subtree for each size-(d+1) subset of $\{R_i\}$
 1. without regard to order of result records
 2. in each “natural” sorted order of result records
- **Use results from child node**
- **Include pipelining strategy**
- Remember best plans and pipelining strategy for each subset
 - can reconstruct order going back down tree
- **Running time exponential** in k
 - still consider each subset of $\{R_i\}$
 - don't consider each order of R_i 's at next level

23

Other operations

- Move selects and projects up/down tree
- Try to **push selects** down tree
 - Pushing **projects** can also be useful
 - why?
 - not always good idea: destroys indexes
- can include in left-join-tree analysis
- Text has detailed discussion equivalences for relational algebra operations

24

Index-only Algorithms

If have indexes giving pointers to records for all relations in query, consider:

- Use indexes to execute operations
 - must have right search keys
- Retrieve records only at end
- If need only count, never retrieve full records

25

Summary

- Have seen in detail how to execute joins
- Have considered execution of other relational alg. ops
- Have looked at how estimate sizes of results
- Have briefly considered one heuristic for making plan for several joins
 - restrict to left-deep trees
- Have looked briefly at planning when relational alg. expr. has more than just joins

26