CS143: Query Optimization

Ch 14

1

Basic Steps in Query Processing

1. Parsing and translation:
   … Translation of SQL queries into naïve RA
2. Optimization: of RA expressions
3. Evaluation

2

Optimization of RA Expressions

The relational expression is transformed into an equivalent one that is less expensive to compute. Main trasformations:

- Pushing Selection and Projection into expressions
- Reorder Joins into sequences with minimal cost
- Much more ….

3

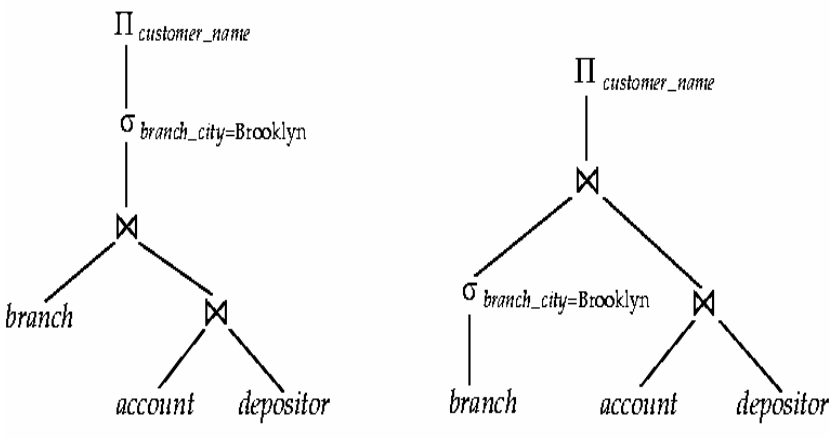
Equivalence rules

- Selections are commutative
- Joins are commutative associative
- Selection and projections distribute over joins ...
- Etc. etc.

4

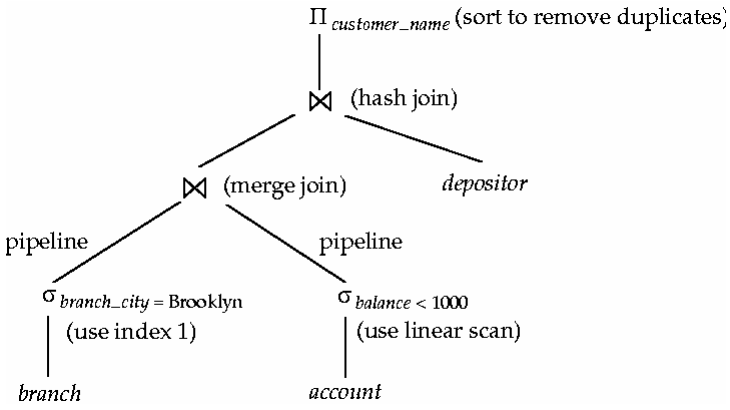
Introduction

- Pushing selection (and also projection)



5

- An **evaluation plan** defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated.



6

Optimization of RA Expressions

- Use the equivalence rules of RA to transform the original ones into others that can be executed more efficiently
- Two modes of optimization
  - **Rule-based or greedy**: When the current expression match a certain pattern transform it in a fixed way.
    - selections on cartesian products becomes joins,
    - Push selection down
  - **Cost Based:**

1. generate expressions logically equivalent to the current one (e.g., all different orders for joins)
2. Estimate the cost of each, and select the best

7

Optimizers for DB Queries vs. PL Compilers

- In compilers optimization is rule-based: when a pattern holds transformation is performed.
  - statements are taken out of nested loops whenever possible
- In DBMS this kind of (greedy) optimization is performed when pushing selection and projection into the the RA expression
- For other operators (joins) the optimization is exhaustive: all possible implementations are evaluated and their cost is estimated
  - Basically, an exponential computation (on the number of joins)
  - Cost estimates requires keeping statistics on DB population
  - Optimality is not guaranteed: but real bad plans are normally avoided

8

Statistical Information for Cost Estimation

- n_r: number of tuples in a relation *r.*
- b_r: number of blocks containing tuples of *r.*
- l_r: size of a tuple of *r.*
- f_r: blocking factor of *r* — i.e., the number of tuples of *r* that fit into one block.
- *V(A, r):* number of distinct values that appear in *r* for attribute *A;* same as the size of $\Pi_A(r)$.

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

9

Cost-Based Optimization

- Cost Measure:
  - Typically disk access is the predominant cost, and is also relatively easy to estimate.
  - Therefore *number of block transfers from disk* is used as a measure of the actual cost of evaluation.
  - It is assumed that all transfers of blocks have the same cost.
- We do not include cost to writing output to disk.
- We refer to the cost estimate of algorithm *A* as E_A

10

Selection Size Estimation

- **Equality selection** $\sigma_{A=v}(r)$
  - *SC(A, r)* : number of records that will satisfy the selection
  - *SC(A, r) can be estimated as r / V(A, r)*
  - $\lceil SC(A, r)/f_r \rceil$ — number of blocks that these records will occupy

Statistical Information for Examples

- $f_{account}$= 20 (20 tuples of *account* fit in one block)
- *V(branch-name, account)* = 50 (50 branches)
- *V(balance, account)* = 500 (500 different *balance* values)
- r = 10000 (*account* has 10,000 tuples)
- Assume the following indices exist on *account:*
  - A primary, B$^+$-tree index for attribute *branch-name*
  - A secondary, B$^+$-tree index for attribute *balance*
- *Estimate the cost of:* $\sigma_{branch\text{-}name = \text{“Perryridge”}}(account)$
  - *V(branch-name,account)* is 50
  - 10000/50 = 200 tuples of the *account* relation pertain to Perryridge branch
  - 200/20 = 10 blocks for these tuples

Selection Cost with no Index

$\sigma_{branch\text{-}name = \text{“Perryridge”}}(account)$

- Number of blocks is $b_{account}$ = 500: 10,000 tuples in the relation; each block holds 20 tuples.
- Linear scan: up to 500 blocks accessed—260 on the average
- Assume *account* is sorted on *branch-name.*
  - A binary search to find the first record would take $\lceil log_2(500) \rceil$ = 9 block accesses
- Total cost of binary search is 9 + 10 -1 = 18 block accesses

Selections Involving Comparisons

- Typical condition is A > costant
- Some RDMSs keeps histograms and other statistics
- In absence of statistical information *c* can be ssumed to be n_r / 2 or n_r / 3
- More complex formulas for conjunctive & disjunctive conditions

Join Ordering

- $r_1 \bowtie r_2 = r_2 \bowtie r_1$: but their cost is not the same
- Join of three relations: r_1, r_2 and r_3

$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$

(Joins are associative and commutative)

- If $r_2 \bowtie r_3$ is quite large and $r_1 \bowtie r_2$ is small, we first compute $(r_1 \bowtie r_2)$ & join the result with r_3
- But if $r_1 \bowtie r_2$ is larger than $r_2 \bowtie r_3$ we compute $(r_2 \bowtie r_3)$ and join the result with r_1
- Also $(r_1 \bowtie r_3)$ first should not be excluded (unless this is actually a cartesian product)

15

Estimation of the Size of Joins

- If $R \cap S$ *in* S is a foreign key in S referencing R, then the number of tuples in $r \bowtie s$ is the same as the number of tuples in s.
- If $R \cap S$ is a key for R, then a tuple of s will join with at most one tuple from r
  - therefore, the number of tuples in $r \bowtie s$ is no greater than the number of tuples in s.
- *V(A,r): the number of distinct values for attribute A in relation r*

16

Estimation of the Size of Joins

- Estimate size of *depositor* $\bowtie$ *customer* using the information about foreign keys:
- *Depositor* has 5000 tuples and *customer* has 10000 tuples

17

Estimation of Size of Joins $R \bowtie S$.

- $R \cap S = \{A\}$ is not a key for R or S. If we assume that every tuple t in R produces tuples in $R \bowtie S$, the number of tuples in $R \bowtie S$ is estimated to be:

$$\frac{n_r * n_s}{V(A,s)}$$

- If the reverse is true, the estimate obtained will be:

$$\frac{n_r * n_s}{V(A,r)}$$

- The lower of these two estimates is probably the more accurate one.
- Can improve on above if histograms are available
  - Use formula similar to above, for each cell of histograms on the two relations

18

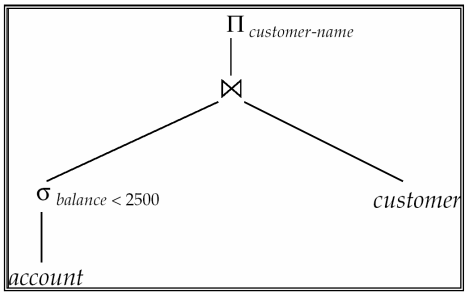
Estimation of the Size of Joins

- Estimate size of *depositor* ⋈ *customer* without using information about foreign keys:
- *Depositor* has 5000 tuples and *customer* has 10000 tuples
  - *V(customer_name, depositor)* = 2500, and *V(customer_name, customer)* = 10000
  - The two estimates are:
    - 5000 * 10000/2500 = 20,000 and
    - 5000 * 10000/10000 = 5000
  - We choose the lower estimate, which in this case, is the same as our earlier computation using foreign keys.

19

Materialization vs Pipelining

- **Materialized evaluation:** evaluate one operation at a time, starting at the lowest-level. Use intermediate results materialized into temporary relations to evaluate next-level operations.
- **Pipelined evaluation:** one tuple at the time.



20

More Optimization

- More ... lots of special techniques—some vendor specific
- Commercial systems are surprisingly smart ... particularly if you listen to their marketing people
- Most of them let the user see and control the optimization plan.

21

Statistics collection commands

- DBMS has to collect statistics on tables/indexes
  - For optimal performance
  - Without stats, DBMS does stupid things…
- DB2
  - RUNSTATS ON TABLE <userid>.<table> AND INDEXES ALL
- Oracle
  - ANALYZE TABLE <table> COMPUTE STATISTICS
  - ANALYZE TABLE <table> ESTIMATE STATISTICS (cheaper than COMPUTE)
- Run the command after major update/index construction

22