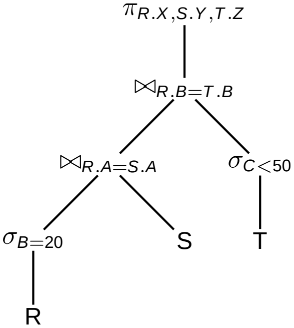
CS5226 Lecture 9
Statistics Tuning

Statistics Tuning Issues

- Which statistics to collect?
- At what level of detail of the statistics to collect?
- When to collect/refresh?
- Challenges
  - outdated statistics
  - improperly configured statistics

Cost Estimation of Query Plan

- Cost estimation involves the following:
  1. What is the evaluation cost of each operation?
     - Depends on: size of input operands, available buffer pages, available indexes, etc.
  2. What is the output size of each operation?



Examples of Statistics

- cardinality (i.e. number of tuples)
- number of distinct values in each column
- the highest & lowest values in each column
- column group statistics
- histograms
- frequent values of some columns

Size Estimation

- Consider a query $q = \sigma_p(e)$
  - $p = t_1 \ \wedge \ t_2 \ \wedge \ \cdots \ \wedge \ t_n$
  - $e = R_1 \times R_2 \times \cdots \times R_m$
- How to estimate $||q||$?
- We have $||e|| = \prod_{i=1}^{m} ||R_i|| = ||R_1|| \times ||R_2|| \times \cdots \times ||R_m||$
- Each term t_i potentially eliminates some tuples in e
- Reduction factor of a term t_i (denoted by rf_i) is the fraction of tuples in e that satisfy t_i; i.e., $rf_i = \frac{||\sigma_{t_i}(e)||}{||e||}$
- Assuming the terms in p are statistically independent,

$$||q|| = ||e|| \times \prod_{i=1}^{n} rf_i = ||e|| \times rf_1 \times rf_2 \times \cdots \times rf_n$$

Estimation assumptions

- **Uniformity assumption**
  - uniform distribution of attribute values
- **Independence assumption**
  - Independent distribution of values in different attributes
- **Inclusion assumption**
  - For a join predicate $R.A = S.B$, we have $(\pi_A(R) \subseteq \pi_B(S))$ or $(\pi_A(R) \supseteq \pi_B(S))$

How to estimate reduction factor?

Form of t_i	Estimation of rf_i
$A_j = v$	$\frac{1}{N_{key}(I)}$ if I_{A_j} exists, $\frac{1}{10}$ otherwise.
$A_j > v$	$\frac{IHigh(I_{A_j})-v}{IHigh(I_{A_j})-ILow(I_{A_j})}$ if I_{A_j} exists, $\frac{1}{10}$ otherwise.
$A_j = A_k$	$\frac{1}{\max\{N_{key}(I_{A_j}),N_{key}(I_{A_k})\}}$ if both I_{A_j} and I_{A_k} exist, $\frac{1}{N_{key}(I_{A_j})}$ if only one index (say I_{A_j}) exists, $\frac{1}{10}$ otherwise.

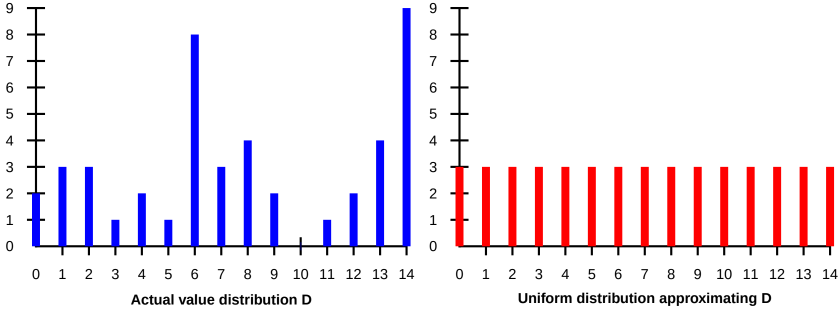
- $N_{key}(I_{A_j})$ = number of keys in index I_{A_j}
- $ILow(I_{A_j})$ = minimum key value in index I_{A_j}
- $IHigh(I_{A_j})$ = maximum key value in index I_{A_j}

Improved Estimation using Histograms

- histogram = statistical information maintained by DBMS to estimate data distribution
- Main idea:
  - partition attribute's domain into sub-ranges called buckets
  - assume distribution within each bucket is uniform
- Types of histograms:
  - Equiwidth histogram: each bucket has equal number of values
  - Equidepth histogram: each bucket has equal number of tuples

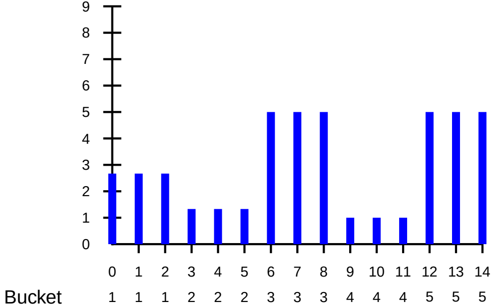
Uniform vs Non-uniform Distributions

- Total number of distinct values = 15
- Total number of tuples = 45
- Uniform distribution assumption: each value has $\frac{45}{15} = 3$ tuples

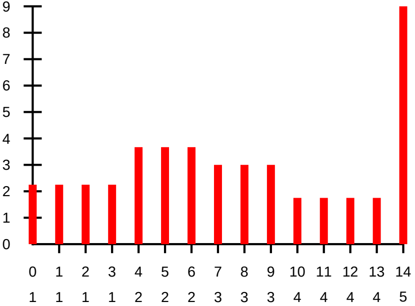


Equiwidth vs Equidepth Histograms

Bucket No	Equiwidth		Equidepth	
	Value Range	No. Tuples	Value Range	No. Tuples
1	[0, 2]	2+3+3=8	[0, 3]	2+3+3+1=9
2	[3, 5]	1+2+1=4	[4, 6]	2+1+8=11
3	[6, 8]	8+3+4=15	[7, 9]	3+4+2=9
4	[9, 11]	2+0+1=3	[10, 13]	0+1+2+4=7
5	[12, 14]	2+4+9=15	[14, 14]	9



Equiwidth Histogram



Equidepth Histogram

Statistics Tuning Approaches

Reactive approaches

- Monitor a query during execution
- Observe errors between estimates and actual values from query feedback
- Possible strategies:
  - Use errors as adjustment factors to correct statistics for future queries,
  - Trigger statistics collection when error exceeds some threshold, or
  - Re-optimize current query

Statistics Tuning Approaches (cont.)

Proactive approaches

- Try to predict, identify and possibly solve potential problems before query execution
- Possible strategies:
    - Monitors update/delete/insert (UDI) activity on data & automatically refreshes statistics when UDI activity exceeds some threshold
    - Analyze queries to determine which statistics to collect
    - Maintains multiple plans for each query & adaptively change plan based on query runtime feedback

Statistical Views (Statviews)

- A statview is a view definition augmented with statistics collected on the result of executing this view, without the actual data

- DB2 Example:

create view sv_cust_order **as** (
  select *
  from Customer C, Order O
  where C.cust# = O.cust#)

alter view sv_cust_order **enable query optimization**

runstats on table DB2DBA.sv_cust_order **with distribution**

Statistical Views (Statviews) (cont.)

Query: **select** *
from Customer C, Order O
where C.cust# = O.cust#
and C.country = ‘Singapore’

- Without statview sv_cust_order,
  - estimate selectivity of selection predicate on C
  - estimate selectivity of join predicate on $C \times O$
  - apply independence assumption to estimate cardinality of result
- With statview sv_cust_order,
  - estimate selectivity of selection predicate on statview

Statview Tuning

- **Input**:
  - a workload $W = \{Q_1, \cdots, Q_n\}$, each Q_i is a query,
  - c_{max}, the maximum number of statviews that can be maintained
- **Output**:
  - A set of at most c_{max} statviews that minimizes the execution time of W

Statview Tuning (cont.)

- Let P denote a query plan
- C = estimated cost of P based on some simplifying assumptions (e.g. uniformity, independence, or inclusion)
- C^{P}_{acc} = accurate cost estimate of P where the cardinality at each operator in P is estimated without any simplifying assumptions
- Cost of P is overestimated if $C > C^{P}_{acc}$
- Cost of P is underestimated if $C < C^{P}_{acc}$

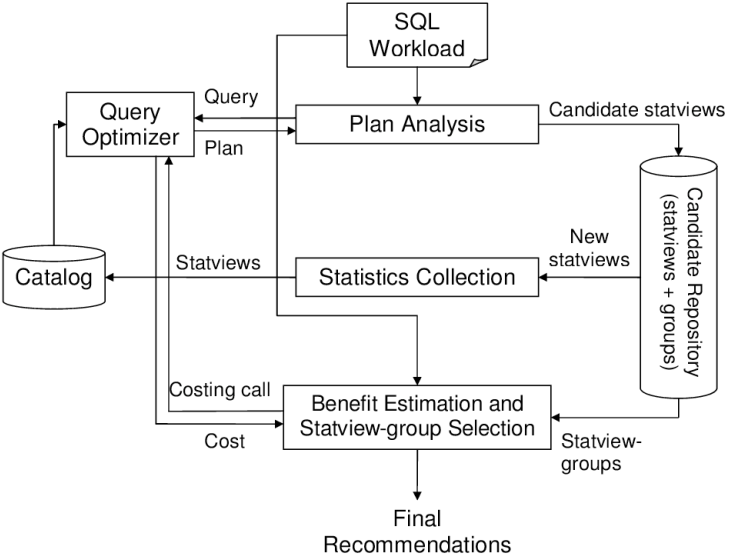
Relevant statviews

- E = subexpression in a plan P for query Q
- V = statview corresponding to E
- V is a relevant statview for optimizing Q if
  1. cost of E is underestimated & P is chosen by optimizer, or
  2. cost of E is overestimated & P will be chosen by optimizer when cost of E is corrected
- First type identified by “optimize & find-statviews” loop
- Second type identified by query structure analysis
  - predicates involving attributes with skewed distributions, arithmetic expressions, or UDFs
  - join predicates that violate inclusion assumption, etc

Statview Tuning

- **Optimization goal**:
  - Collect enough statviews to find optimal plan
- **Revised optimization goal**:
  - Collect enough statviews to find predictable plan
  - Plan cost is not underestimated

StatAdvisor Architecture



StatAdvisor

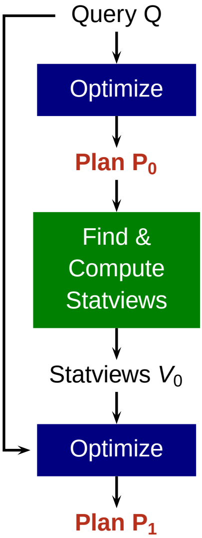
Input: Query workload W & c_{max} constraint
Output: Set of recommended statviews for W

$(V, G) \leftarrow$ PlanAnalysis(W)
while ($V \neq \emptyset$) do
    CollectStatviews(V)
    $(V, G) \leftarrow$ PlanAnalysis(W)
EstimateBenefit(G)
$R \leftarrow$ StatviewGroupSelection(G, c_{max})
return R

How to collect statistics for a statview?

- Statview V is on a single table R
  - Create a random sample S of R
  - Scan S to compute statistics for V
- Statview V is on multiple joined tables $R_1 \bowtie \cdots \bowtie R_n$
  - Let R_1 be the root table
    - ★ R_1 has a foreign key to each of the tables $R_2, \cdots, R_n$
    - ★ Each of the tables $R_2, \cdots, R_n$ has no foreign key to R_1
  - Compute the join synopsis J for the root table R_1
    1. Create a uniform random sample S of R_1
    2. For each table R_i that R_1 has a foreign key to, join S with R_i
    3. Repeat step 2 recursively
  - Scan J to compute statistics for V

Enumeration of candidate statviews



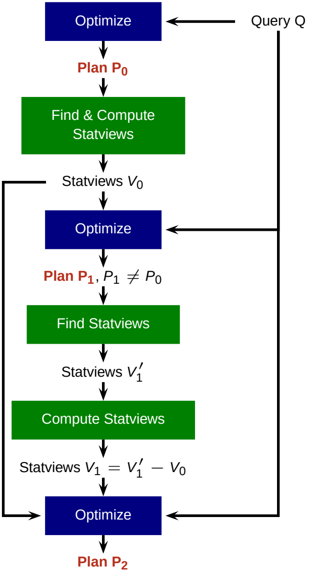
Case 1: $\mathbf{P_1 = P_0}$

- V_0 is not beneficial for optimizing Q

Case 2: $\mathbf{P_1 \neq P_0}$

- Either cost of P_1 is underestimated & P_0 could be a better plan than P_1
- Or P_1 is indeed a better plan than P_0

Enumeration of candidate statviews



Case 1: $P_2 = P_0$

- V_1 corrected the underestimation of cost of P_1

Case 2: $P_2 = P_1$

- V_1 is not beneficial for optimizing Q

Case 3: $(P_2 \neq P_0)$ and $(P_2 \neq P_1)$

- Either cost of P_2 is underestimated & P_0/P_1 could be a better plan than P_2
- Or P_2 is indeed a better plan than P_0 & P_1

Enumeration algorithm for query

Input: Query Q

Output: Candidate statviews for Q

$V \leftarrow \emptyset$; stop $\leftarrow$ *false*; $k \leftarrow 0$;
repeat
    generate the optimal plan P_k for Q using statviews V
    find the important statviews VT for plan P_k
    $VT \leftarrow VT - V$
    if ($VT = \emptyset$) or ($P_k = P_j$ for some $j < k$) then
        stop $\leftarrow$ *true*
    else
        compute statviews VT
        $V \leftarrow V \cup VT$
    $k \leftarrow k + 1$
until (stop)
return V // V is a statview-group for Q

Enumeration algorithm for query (cont.)

- Let P be optimal plan for Q using V
  - V = the set of returned statviews
- P is guaranteed to be a predictable plan
- The approach could miss a better plan P' if
  1. the cost of some subexpression E of P' is overestimated, and
  2. E is not a subexpression of any enumerated plan

Enumeration algorithm for workload

Input: Workload $W = \{Q_1, \cdots, Q_n\}$
Output: Candidate statviews for W

for each query $Q_i \in W$ do $V_i \leftarrow \emptyset$
$k \leftarrow 0$
repeat
> for each query $Q_i \in W$ do
>> generate the optimal plan $P_{i,k}$ for each Q_i using V_i
>> find the important statviews VT_i for $P_{i,k}$
>> $VT_i \leftarrow VT_i - V_i$
>> if ($VT_i = \emptyset$) or ($P_{i,k} = P_{i,j}$ for some $j < k$) then
>>> remove Q_i from W
>>
>> else
>>> $V_i \leftarrow V_i \cup VT_i$
>
> partition $\{VT_j \mid Q_j \in W\}$ into batches $\{B_1, \cdots, B_m\}$
> compute statviews for each batch B_i
> $k \leftarrow k + 1$

until (W is empty)
return $(V_1 \cup \cdots \cup V_n)$

Benefit estimation

- P = optimal plan without statviews
- P_V = optimal plan with statviews V
- $\mathbf{B(V, Q)}$ = benefit of statviews V for query Q
  - $B(V, Q) = ActCost(Q, P) - ActCost(Q, P_V)$
  - $ActCost(Q, P)$ = actual execution cost of query Q using plan P
- $\mathbf{B'(V, Q)}$ = estimated benefit of statviews V for query Q
  - $B'(V, Q) = EstCost(P, V) - EstCost(P_V, V)$
  - $EstCost(P, V)$ = estimated execution cost of plan P with statviews V
  - Why not $EstCost(P, \emptyset) - EstCost(P_V, V)$?

Benefit estimation (cont.)

$\mathbf{B}(\mathbf{V}, \mathbf{W})$ = benefit of V for a workload W

- $W_V = \{Q \in W \mid \text{Q generated V}\}$
- $B(V, W) = \sum_{Q \in W_V} B'(V, Q)$

Statview-group selection

Input: Set of statview-groups G & constraint c_{max}

Output: Set of recommended statview-groups R

```
c ← 0;  R ← ∅
while (|G| > 0) and (c < c_max) do
    V_best ← null
    B_best ← 0
    for each V ∈ G do
        if (B(V) > B_best) and (C(V, R) ≤ c_max − c) then
            V_best ← V
            B_best ← B(V)
    if (V_best ≠ null) then
        c ← c + C(V_best, R)
        G ← G − V_best
        R ← R ∪ V_best
    else
        break
return R
```

References

Required Readings

- A. El-Helw, I.F. Ihab, and C. Zuzarte, *StatAdvisor: recommending statistical views*, VLDB 2009

Additional Readings

- C. A. Galindo-Legaria, M. Joshi, F. Waas, and M.-C. Wu, Statistics on Views, VLDB 2003.
- A. Aboulnaga, P. Haas, M. Kandil, S. Lightstone, G. Lohman, V. Markl, I. Popivanov, V. Raman, *Automated statistics collection in DB2 UDB*, VLDB 2004.