**Deep Bodra**

**5801 1841**

**DBI Week 8(3/30 - 4/3)**

**Detailed explanation of the topics covered in video lecture**

**Video No 35**

1. Estimation
   1. A query can be executed in multiple ways as there are many choices like use a sequential search or a B+ tree
   2. For very large result sequential search might be faster and for smaller results a B+ tree might be faster
   3. So there is a need to make a choice which depends on how large the result would be for a given query
   4. Selectivity / Cardinality
      1. How many tuples will be produced as a result of a given selection query?
      2. Case 1: Selection attribute is a key
         1. One tuple or 0 tuple
      3. Case 2: Selection attribute is other than key
         1. Domain
            1. Information of range of values for the selection attribute
            2. This is not so helpful
         2. Domain + Uniformity
            1. The tuples are distributed uniformly
            2. This is helpful but the uniformity is very rare in practical data
            3. eg, n(Domain) = 1024

Fraction f = 1/1024 for (select \* from R where a=3)

* + - 1. Histogram
         1. Generate a histogram based on attribute values by grouping tuples with a particular range of values into respective buckets
         2. Also, maintain count/bucket (number of tuples for each bucket)
         3. Equi width

Uniformly distribute the tuples in the available buckets

For a given query with predicate having literal x, calculate the bucket number and then use groupby to find the count

In this the error in estimation is dependent on the number of the tuples in the bucket which has the requested tuple

This is writable in SQL

Drawback: For distribution like skew it gives equal importance to all

* + - * 1. Equi depth

Instead of equal width we have same count for each of the buckets

This is better than the previous one

This is not writable in SQL

* + - * 1. m biased histograms

The tuples with top m frequent domain values are given more importance by grouping them in “n-1” buckets

All the remaining tuples go in the nth bucket

This is better than the previous

* + - * 1. Multi dimensional histograms

All the above methods can be implemented for multi dimensional predicates

Drawback: the histogram size may turn in the order of the original data or even more

Solution: Assume the dimensions(attributes) are independent

Eg, for relation R(A,B,C,..)

|select(A=x, B=y)| = probability(A=x) \* probability(B=y) \* |R|

* 1. Join
     1. Case 1: Primary and Foreign key join
        1. |R join S| = Summation frequency(R, i) \* frequency(S, i) where i belongs to Domain
        2. Estimation can be done by developing histograms and using the counts of the buckets instead
  2. Query Optimization
     1. Given 2 plans which will run faster?
     2. How do you know that without actually running the query?
     3. Well, you can’t but you can predict/estimate.
     4. This estimation is dependent on the cardinality of the operations in the plans
     5. However, the correct answer is very loosely coupled with the cardinality. (Linear scans may be faster than random scans)
     6. We could also depend on the previous results
     7. But that is pretty much it. There is no way for sure to be correct always

**Video No 36**

1. Query Cost Estimation
   1. Given a plan estimate the cost of running
   2. Generally, Cost function = no of blocks to process (considering all the tuples throughout the query plan)
   3. B(R) = no of blocks; T(R) = number of tuples
   4. Goals:
      1. Relative order matters
         1. Of the 2 plans the plan with higher execution time must have higher cost
         2. If the difference in the execution time is significant then it should be reflected in the cost of the plans
      2. If costs of the 2 plans are in x%, then any plan is good to go
2. Query Optimization
   1. It is not pragmatic to run though all the possible execution plans
   2. Optimization heuristics to cut down prospective plans
      1. Push the selection down if possible
      2. Projection requires no optimization as such
      3. Order of joins is the biggest problem
         1. Most DB’s consider joins in the form of left deep plans only. Even this has an exponential number of plans. Dynamic programming can be used to to estimate the cost in polynomial time
         2. Don’t evaluate plans for smaller joins
         3. Push the smaller/medium relations at the bottom right of the plan (smaller meaning that fit in cache)
      4. There are errors in estimation, cost function estimation, etc. These errors stack up and the DB may end up taking the plan which might be very bad.
   3. Database Tuning
      1. If optimization is not good, then using extra statistics or hacks to make it more reliable for certain sets of queries is called database tuning
   4. Adaptive Execution
      1. A rough plan is created at first and executed
      2. In the middle of the execution based on the performance the plan is changed on the fly without affecting the correctness of the query

**Video No 37**

* 1. Selectivity for disjunctions
     1. |c1 v c2| = |c1| + |c2| - |c1 ^ c2| (|c1 ^ c2| can be computed using independent assumption
     2. This can be extended for n literals
     3. Alternatively, we can ignore the negative conjunctions in the formula which would increase the cost and wouldn’t harm our query plan selection
  2. Map Reduce (continued)
     1. High level process
        1. Read block (usually 64MB)
        2. Map block
        3. Combine block
        4. Distribution + Partitioning
           1. Problem: Given (k, v) pairs. Group based on hash of the keys k
           2. Tell the controller that the group is ready to be given to the reducer
           3. Send/Pull the groups to/from the other nodes through/over the network
     2. GroupBy
        1. 10-100B tuples, 1-2M tuples
        2. Writing map reduce
           1. Map = identity (do nothing)
           2. Combiner works on a block. After the combiner has finished it will have lots of tuples but these tuples will be distributed across various groups. This means that each group will have very few tuples (2-3). As a result large amount of data will have to be transferred over the network which would become a bottleneck.
           3. Reducer will have to wait a lot to gather the data they need
        3. Flexibility and power of GLA’s
           1. After the computation of the states is done in the aggregated tree fashion, the states in different machines can also be added in the aggregated fashion
           2. How does this overcome network bottleneck?

The left half of the aggregation tree will be given to one rack of servers(all connected through the same switch) and the right half to another rack

The switches of the racks are also connected. The data required for the last aggregation will be transferred over this switch

--THANK YOU--