Yifei Huang
Redbase Extension Architecture

General Algorithm for Query Optimizer:

At the core of the extension is the dynamic programming algorithm. Like QL, I will apply selection as soon as possible (i.e. if there is a condition on R.A and S.A, then I will apply it as soon as I have joined R and A). We consider the following pseudocode:

$R \leftarrow$ set of relations to join (e.g., ABCD)
For ∂ in {1...|R|}:
    for S in {all length ∂ subsets of R}:
        optcost(S) = argmin_a (a join (S-a) + **optcost**(S-a))

The optcost is calculated two different ways based on whether the join should be an index join or a nested loop join:
Index: (S-a).cost + (S-a).numTuples * $\frac{a.numTuples}{a.joinAttr.numDistinct}$
Nested: (S-a).cost + (S-a).numTuples ($\frac{a.numTuples * a.tupleSize}{PAGE\ SIZE}$)

Keeping track of Statistics:

We extended the information stored about each attribute and relation. I will create two additional fields to each relcat entry:

1. numTuples - the total # of attributes in the relation
2. statsInitiated - whether the statistics for that relation have been initialized

The entries in attrcat will also be extended by three fields:

1. numDistinct – total number of distinct values
2. maxValue – maximum value (converted to float)
3. minValue – min value (converted to float)

These relation statistics will be calculated upon initial load, and never again until specified by user input by typing "set calcStats="<relation name>"". To print the statistics for a relation, enter "set printStats="<relation name>" ".

Computing inequality for strings becomes a bit more complicated. For this, we use the first char of the string (we are guaranteed strings of at least length 1) and convert it to a float. It's still more than 26 values because of capitals/lower cases/special characters.

To keep track of the statistics during the dynamic programming, we create an array of maps, which I call optcost. Optcost[i] refers to a map for joins of size i+1. Each of these maps maps from join sets to statistics values.

The join sets are represented by an int bitstring. The relations are assigned an index based on the order they arrive in the select statement. The corresponding bit is then set if the relation is in the join set. For example, the key value of ….01011 = 11 refers to a set containing the 1st, 2nd and 4th relations.

The value in each map is a costElem struct, containing. For set S, it contains:

- Int joins – bits referring to relations in S-a
- Int newRel – join # given to relation a
- Float numTuples – estimated # of tuples in (S-a)
- Float cost – estimated cost of joining all tuples in S
- Int indexAttr/indexCond – attribute and condition to indexscan/join on
- Map attrStats – map from attribute index to its stats

Each attrStat contains:

- Int numDistinct – the number of distinct values of this attribute
- Float maxValue – max value for this attribute
- Float minValue – min value for this attribute

Recalculating stats after applying conditions:

Nonequality:
Because we are estimating the # of tuples, we do not need to be exact. We will make the assumption that the domain of each attribute is large, and that it is unlikely to find an equality condition. Therefore, we assume the worst case scenario that we do not get a match, so all tuples satisfy the inequality and we propagate them all up.

Equality and Inequality:
I used the preservation of value sets, containment of value sets, assumed that attribute values are distributed evenly. After applying each condition I did a normalization step for each attribute B not affected by the condition:
V(B) = min(V.B, new total # of tuples)
This is because we can't have more distinct values of a tuple than the total # of tuples in a relation.
I also do a sort of "smoothening" for the inequality cases if ever the range of an attribute is 0. I add 1 to the numerator, and 2 to the denominator, so if both the new range and old range is 0, we assume that half of the tuples pass the inequality condition.
See appendix for detailed algorithms for these (given with ex_DOC during demo)

Other notes:
To disable and enable QO, do "set useQO="true"" or "set useQO="false"".
To enable or disable the printing of the GetPage stats, do "set printPageStats="true"" or "set printPageStats="false"".

Testing strategy:
I tested my queries first for correctness, then for optimizations. First, I ran the system again on the same queries I used to test in the QL component, and the queries used in the competition, since the outputs for both are known, and I had ground truth to compare to. Afterwards, I compared the GETPAGE value for each of the queries, with and without the optimizer. For the most part, especially for larger tables, the QO worked well, which is reassuring. I also wrote a function, PrintRels, to print all the contents of optjoin.

Appendix:

Notation:
t(R) – total number of tuples in R
V(R.A) – total number of distinct attributes of R.A
Vmin(R.A) – min value of R.A
Vmax(R.A) – max value of R.A
Use attribute B to denote all attributes not in the condition

Equality condition
Selection on R.A=a
- Total # of tuples = t(R)/V(R.A)
- V(R.A) = 1
- Vmin(R.A)=Vmax(R.A) = a
- V(R.B) = min(V(R.B) , new total # of tuples)

Selection on R.A = S.A
- Total # of tuples = t(R)xt(S)/(max{R.A,S.A})
- V(R.A) = min(R.A, S.A)
- V(S.A) = min(R.A, S.A)
- V(R.B) = min(V(R.B) , new total # of tuples)

Inequality conditions:
R.A < value or R.A <= value
- Total # of tuples = t(R)*(value-Vmin(R.A)+1)/(Vmax(R.A)-Vmin(R.A) + 2)
- V(R.A) = R.A*(value-Vmin(R.A)+1)/(Vmax(R.A)-Vmin(R.A) + 2)
- V(R.B) = V(R.B) stay the same for all other attributes B in R
- Vax(R.A) = Min(value, Vmax(R.A))

R.A < S.A or R.A <= S.A
- Let mid = $\frac{\text{MIN}(\text{Vmax(R.A)},\text{Vmax(S.A)})+ \text{MAX}(\text{Vmin(R.A)},\text{V min(S.A)}+2)}{2}$
- Fraction-S = max((Vmax(S.A) – mid + 1)/(Vmax(S.A) – Vmin(S.A) + 2) , 0.0)
- Fraction-R = max((mid – Vmin(R.A) + 1) / (Vmax(R.A) – Vmin(R.A) + 2) , 0.0)
- Total # of tuples = t(R)*t(S)*MAX(Fraction-S, Fraction-R)
- Max_new(R.A) = MIN(Vmax(R.A),Vmax(S.A))
- Min_new(S.A) = MAX(Vmin(R.A),Vmin(S.A))
- V(R.A) = ((Max_new(R.A) – Vmin(R.A))/(Vmax(R.A) – Vmin(R.A))
- V(S.A) = ((Vmax(S.A) – Min_new(S.A))/(Vmax(S.A) – Vmin(S.A))
- V(R.B) = min(V(R.B) , new total # of tuples)