Database Systems Architecture Project

Range type data

- Range type data contains upper bound and lower bound.
- << only compares the upper bound of one with the lower bound of another.
- && compares the upper bound of one with the lower bound of another, and then does the opposite.
- Therefore, intuitively we want to store these two numbers into histogram.

Planner’s Statistic

- estimate rows to make better query plans
- pg_class tables stores -> rows, pages, histogram of each table
- vacuum analyze -> update the pg_class tables
- planner get no. of pages -> check pages in pg_class col
- if not equal
  estimate the rows to actual table rows
- default histogram bins 100
  - can increase -> but spatially and computationally expensive
- no. of rows = selectivity * relative cardinality(rows in pg_class table)

Histogram

- Range type data includes upper value, and lower value, and its length.
- Upper bound histogram : A equi-depth histogram records each upper of each sorted data.
- Lower bound histogram : A equi-depth histogram records each lower bound of each sorted data.
- Assume the data distribution in each bin follows a uniform distribution.

Illustration

Sort and generate upper histogram

Sort and generate lower histogram

Code

```
delta = (non_empty_cnt - 1) / (num_hist - 1);
deltafrac = (non_empty_cnt - 1) % (num_hist - 1);
pos = posfrac = 0;

for (i = 0; i < num_hist; i++)
{
    bound_hist_values[i] = PointerGetDatum(range_serialize(typcache,
                                                           &lowers[pos],
                                                           &uppers[pos],
                                                           false));
    pos += delta;
    posfrac += deltafrac;
    if (posfrac >= (num_hist - 1))
    {
        /* fractional part exceeds 1, carry to integer part */
        pos++;
        posfrac -= (num_hist - 1);
    }
}
```

calc_hist_selectivity_scalar

- Input a value and a histogram, return % of data that is smaller than that value.
- Error comes from linear interpolation to calculate the selectivity within a single bin which the value located.
- Maximum error = 1/number of bins. (maximum error 1% for 100 bins histogram)

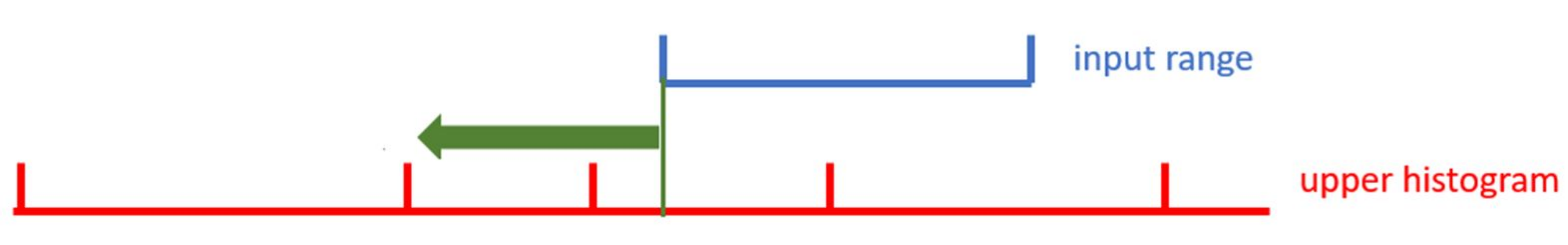
```
index = rbound_bsearch(typcache, constbound, hist, hist_nvalues, equal);
selec = (Selectivity) (Max(index, 0)) / (Selectivity) (hist_nvalues - 1);

/* Adjust using linear interpolation within the bin */
if (index >= 0 && index < hist_nvalues - 1)
    selec += get_position(typcache, constbound, &hist[index],
                          &hist[index + 1]) / (Selectivity) (hist_nvalues - 1);

return selec;
```

Selectivity (<<)

Compare lower bound of input range with upper bound histogram, and get the selectivity scalar, it can be done with a function.

```
case OID_RANGE_LEFT_OP:
    /* var << const when upper(var) < lower(const) */
    hist_selec =
        calc_hist_selectivity_scalar(typcache, &const_lower,
                                     hist_upper, nhist, false);
    break;
```

Illustration



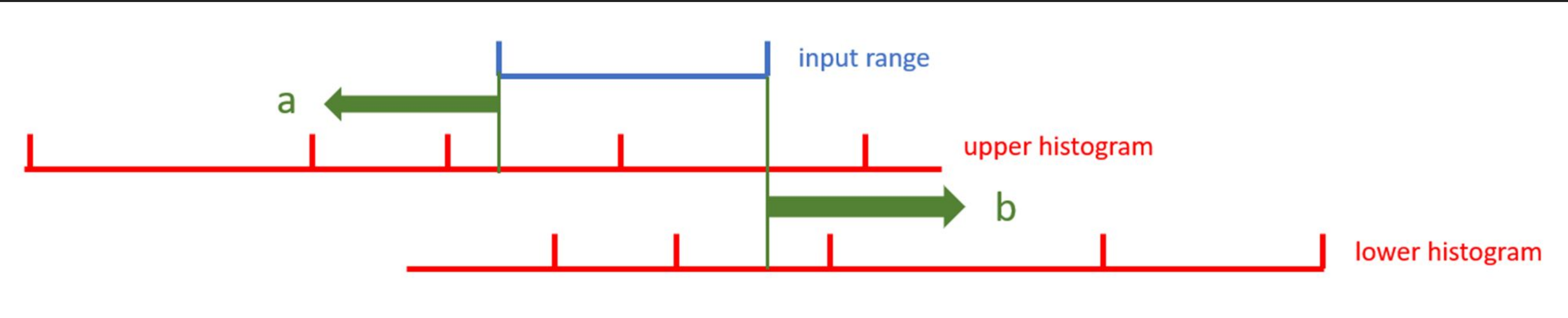
Selectivity (&&)

- 1-(<<)-(>>)
- Two functions calls, maximum 2% error.

```
hist_selec =
    calc_hist_selectivity_scalar(typcache, &const_lower, hist_upper,
                                 nhist, false);
hist_selec +=
    (1.0 - calc_hist_selectivity_scalar(typcache, &const_upper, hist_lower,
                                        nhist, true));
hist_selec = 1.0 - hist_selec;
break;
```

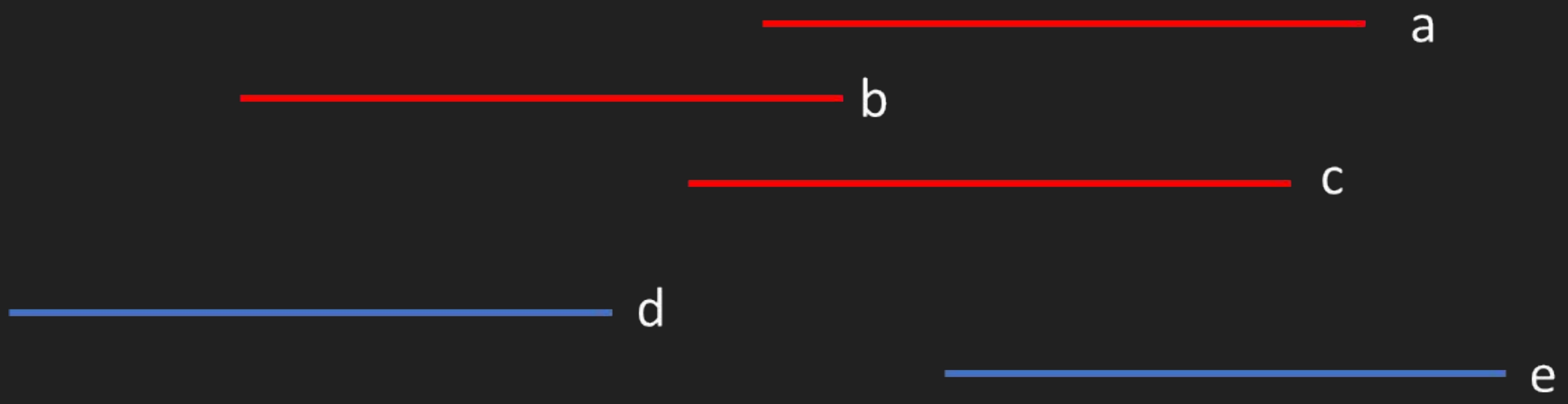
Illustration

Selectivity = 1-a-b

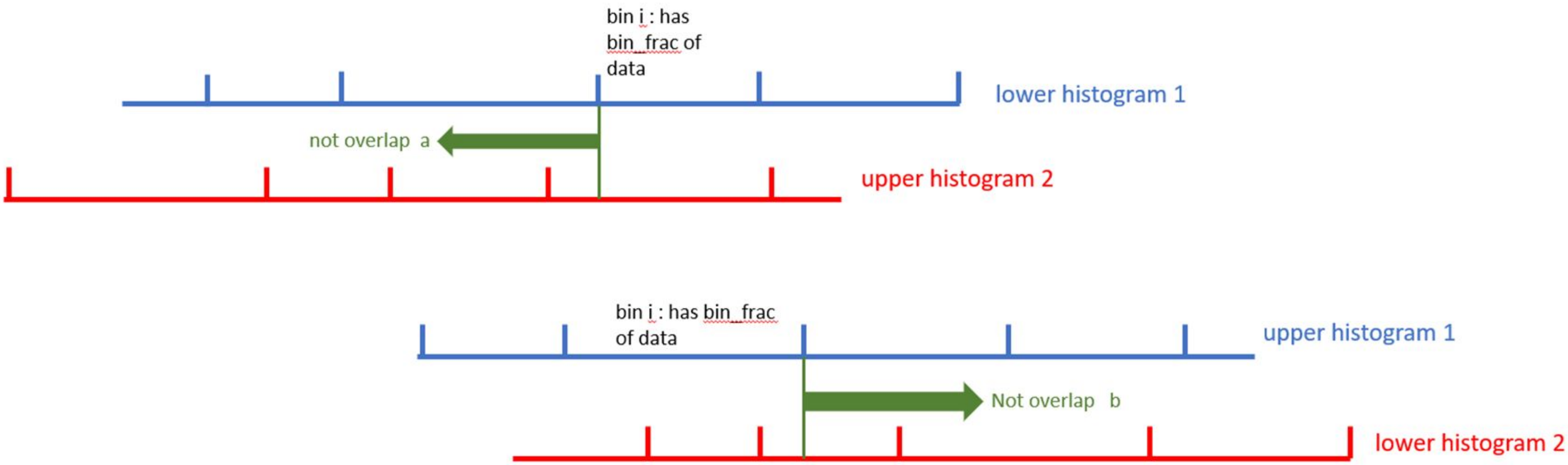


Join cardinality

- Table1 && Table2
- Select all the combine rows that are overlapped from two tables
- (a,e), (c,e), (b,d) will be selected



Illustration



pseudo code

```
bin_frac = 1/number of bins

selectivity = 1.0

For i in every bin of histogram 1:

    lower = lower bound of i-th bin in histogram 1

    upper = upper bound of i-th bin in histogram 1

    a = calc_hist_selectivity_scalar(lower, upper histogram 2)

    b = 1 - calc_hist_selectivity_scalar(upper, lower histogram 2)

    selectivity -= (a+b)*bin_frac
```

Code

- Two functions calls, maximum 2% error.

```
double bin_frac = 1.0/nhist1;
selec = 1.0;
printf("bin_frac:%lf\n",bin_frac);

for (i = 0; i < nhist1; i++)
{
    double a,b;

    RangeBound bin_lower,bin_upper;
    bin_lower.inclusive = true;
    bin_lower.val = (Datum)(hist_lower1[i].val);
    bin_lower.infinite = false;
    bin_lower.lower = true;
    bin_upper.inclusive = true;
    bin_upper.val = (Datum)(hist_upper1[i].val);
    bin_upper.infinite = false;
    bin_upper.lower = false;

    //printf("%d a:%d  b:%d \n",i,bin_lower.val,bin_upper.val);

    a = calc_hist_selectivity_scalar(typcache2, &bin_lower, hist_upper2, nhist2, false);
    b = 1 - calc_hist_selectivity_scalar(typcache2, &bin_upper, hist_lower2, nhist2, true);

    //printf("a:%lf  b:%lf \n",a,b);

    selec -= (a+b)*bin_frac;

}
```

Benchmark data

- All data is between ±500000, 30000 rows
- Random : randomly selected two bounds
- Short length : the length of all data is less than 10000(a bin width)
- Long length : the length of all data is bigger than 100000 (ten bins width)
- Small table : random 300 rows
- Big table : random 300000 rows

Benchmark 1 (<<)

- Select Int4range(-150000,150000)
- Random: 0% error
- Short: 0% error
- Long: 0.01% error
- Small: 0% error
- Big: 0.36% error

```
 Seq Scan on random1 t  (cost=0.00..538.00 rows=3706 width=1
4) (actual time=0.018..6.904 rows=3706 loops=1)
   Filter: (r << '[-150000,150000)'::int4range)
   Rows Removed by Filter: 26294
 Planning Time: 0.258 ms
 Execution Time: 7.041 ms
(5 rows)
```

Benchmark 2 (&&)

- Select Int4range(-150000,150000)
- Random: 0.01% error
- Short: 0.007% error
- Long: 0.013% error
- Small: 0% error
- Big: 0.39% error

```
 Seq Scan on random1 t  (cost=0.00..538.00 rows=22660 width=
14) (actual time=0.009..5.247 rows=22663 loops=1)
   Filter: (r && '[-150000,150000)'::int4range)
   Rows Removed by Filter: 7337
 Planning Time: 0.083 ms
 Execution Time: 6.327 ms
(5 rows)
```

Benchmark 3 (join)

- Random && Random : 0.78% error
- Short && Short : 0.0002% error
- Long && Long : 0. 7% error
- Big && Small : 0.59% error
- Small && Big : 0.51% error

```
 Nested Loop  (cost=0.00..13501001.00 rows=592985948 width=2
8) (actual time=0.636..191153.958 rows=600015905 loops=1)
   Join Filter: (t1.r && t2.r)
   Rows Removed by Join Filter: 299984095
   ->  Seq Scan on random1 t1  (cost=0.00..463.00 rows=30000
 width=14) (actual time=0.332..46.572 rows=30000 loops=1)
   ->  Materialize  (cost=0.00..613.00 rows=30000 width=14)
(actual time=0.000..1.291 rows=30000 loops=30000)
         ->  Seq Scan on random2 t2  (cost=0.00..463.00 rows
=30000 width=14) (actual time=0.295..10.071 rows=30000 loops
=1)
 Planning Time: 3.330 ms
 Execution Time: 207779.222 ms
(8 rows)
```

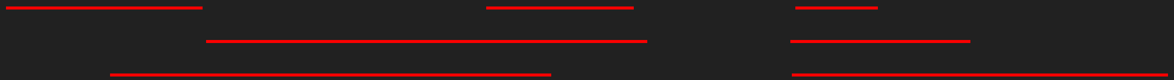
Results

- The error percentage of all results is smaller than 1%
- All the planning time is less than 3.5 milliseconds.
- The only error using these histograms comes from the estimating error when using linear interpolation to calculate the selectivity within a single bin. (uniform distribution assumption)
- All planning times are less than 0.002% of the entire execution time.

Frequency approach theory

This approach is based on the frequency of ranges, meaning that the collected statistics are composed of an equi-width histogram where each bin contains the number of ranges overlapping that bin.

R1



1	2	2	2	2	3	2	0	3	2	1	1

Similarly, the second relation has a range-type column with 7 tuples represented as blue lines.

R2

1	2	1	0	2	0	0	1	2	2	1	1

The selectivity for the strictly left operator with a range [a, b] can be computed by counting the frequencies in all the bins [bin_start, bin_end] where bin_end < a, because all the ranges that intersect with these bins are necessary before the range [a, b]. In any case, the number obtained with this counting is taking into account the same ranges more times (each of them contributes to the frequency of all the bins that overlap), therefore it must be normalized dividing for the sum of the frequency of all the bins. The resulting value should represent an estimation of the number of selected rows.

In the example below, we consider the same histograms as before supposing that the starting point is the same for simplicity.

1	2	2	2	2	3	2	0	3	2	1	1

1	2	1	0	2	0	0	1	2	2	1	1

In this case, the total number of overlapping ranges is :

1+2+4+2+2+4+4+3+4+4+3+3 = 36 tuples

Of course, also in this case this number must be normalized dividing it for the product between the sum of the frequencies of all the bins.

Implementation of frequency approach - typanalyze

```
start_hist = min(lower_bounds);
end_hist = max(upper_bounds);
bin_width = (end_hist - start_hist) / num_hist;

histogram <- array
for (i = 0; i < num_hist; i++) {
   bin_start = start_hist + bin_width*i
   bin_end = bin_start + bin_width
   frequency = 0
   for (range in column) {
       // if overlaps
       if (range.lower_bound <= bin_end && range.upper_bound >= bin_start) {
           frequency++
       }
   }
   histogram[i] = frequency
}
```

Implementation of frequency approach - selectivity estim.

```
selectivity(A&&B) = 1 - [selectivity(A<<B) + selectivity(A>>B)]
```

Strictly left

```
index, count = 0

while (hist_start + bin_width*(index+1) < constval.lower_bound && index<nbins) {
   count += histogram[index]
   index += 1
}
```

Strictly right

```
index, count= 0

while (hist_start + bin_width*(nbins-index) > constval.upper_bound && index<nbins){
   count += histogram[nbins-index-1]
   index += 1
}
```

Implementation of frequency approach - join estimation

```
count, i1, i2 =  0
bin_start1 = hist_start1
bin_start2 = hist_start2

while  (i1<nbins1 && i2<nbins2) {
   bin_end2 = bin_start2 + bin_width2
   if (bin_end2<=bin_start1) {
       bin_start2 = bin_end2
       i2 +=  1
       continue
   }
   bin_end1 = bin_start1 + bin_width1;
   if (bin_end1 <= bin_start2) {
       bin_start1 = bin_end1
       i1 +=  1
       continue
   }

   count +=  histogram1 [i1] *  histogram2 [i2];
   if (bin_end1 < bin_end2) {
       bin_start1 = bin_end1
       i1 +=  1
   }  else  if (bin_end1 == bin_end2) {
       bin_start1 = bin_end1
       i1 +=  1
       bin_start2 = bin_end2
       i2 +=  1
   }  else  {
       bin_start2 = bin_end2
       i2 +=  1
   }
}
```

Results

Error percentage for different tables and different tested estimation functions

	Random	Short	Long	Small	Big
Selectivity (<<)	26.72%	0.09%	27.02%	28%	27%
Selectivity (&&)	53.89%	0.87%	57.04%	57.67%	54.27%
Join (&&)	64.47%	0.51%	68.56%	66%	64.6%