Lab 3 Write-Up

Jordi Burbano (204 076 325), Keisuke Daimon (604 547 017)

13 March 2015

I. LIFE OF A QUERY IN THE SIMPLEDB SYSTEM

simpledb.Parser.handleQueryStatement() generates the logical plan lp using parseQueryLogicalPlan. First, the parser walks through the tables in the FROM clause, adding a Scan node to lp for each one. Next, it parses the WHERE clause, creating Filter and Join nodes as needed. It then looks for GROUP BY fields. After that, the parser walks the SELECT list, picks out aggregates, and checks for query validity. Finally, for a ORDER BY clause, it adds an ORDER BY expression to lp in the specified order on the specified field.

handleQueryStatement() then creates a physical plan for the query via a call to lp.physicalPlan().Initially, for each table, a SeqScan is initialized and the filter selectivity is initialized to 1. Then, to handle the WHERE clause, for each LogicalFilterNode, the corresponding filter selectivity is scaled by the selectivity computed from the relevant TableStats object calling estimateSelectivity(), which in turn uses an IntHistogram or a StringHistogram, depending on the field's type, to compute the selectivity.

II. DESIGN DECISIONS

Selectivity estimation is implemented in the IntHistogram class, which makes use of an int[][] histogram hist to estimate different types of selectivities. The histogram double array's first dimension is along individual bins while the second dimension records two numbers: the bin's rightmost value and the bin's count. hist[0] corresponds to a special case in that it is used to keep track of the total number of tuples.

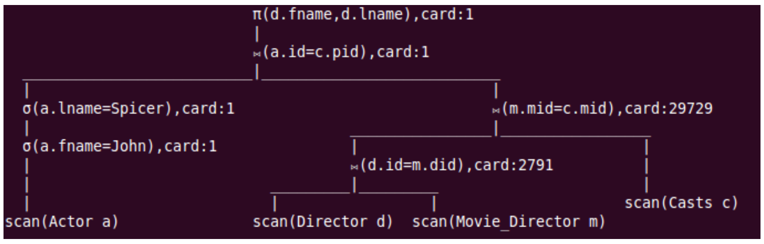
JoinOptimizer.orderJoins() implements the Selinger optimizer. This method makes use of the provided enumerateSubsets() method in considering each subset with size i, as well as the computeCostAndCardOfSubplan to compute each join cost and cardinality.

III. EXAMPLE QUERY PLANS

The following query was executed:

select d.fname, d.lname
from Actor a, Casts c, `Movie_Director` m, Director d
where a.id=c.pid and c.mid=m.mid and m.did=d.id and a.fname='John' and a.lname='Spicer';

The following query plan was generated:



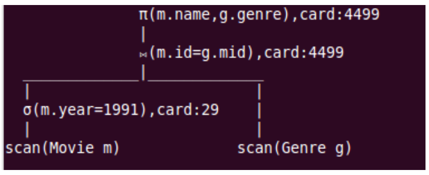
```
                            π(d.fname,d.lname),card:1
                            |
                            ⋈(a.id=c.pid),card:1
  __________________________|___________________________
  |                                                     |
  σ(a.lname=Spicer),card:1                             ⋈(m.mid=c.mid),card:29729
  |                                    _________________|_________________
  σ(a.fname=John),card:1               |                                 |
  |                                    ⋈(d.id=m.did),card:2791           |
  |                           _________|__________                       |
  |                           |                  |                    scan(Casts c)
scan(Actor a)               scan(Director d)  scan(Movie_Director m)
```

In this counterintuitive plan, the result of applying two selections on the relation Actor is joined last perhaps because the optimizer erroneously only considers the initial scan cost of Actor, which is the largest of all relations.

The following query was also executed:

select m.name, g.genre
from Movie m, Genre g
where m.year=1991 and m.id=g.mid;

The following query plan was then generated:



```
               π(m.name,g.genre),card:4499
               |
               ⋈(m.id=g.mid),card:4499
  _____________|_____________
  |                         |
  σ(m.year=1991),card:29    |
  |                         |
scan(Movie m)             scan(Genre g)
```

In this plan, as one would expect, the result of applying a selection on the relation Movie is conditionally joined with Genre g, as opposed to performing the join earlier and the selection later.

IV. Changes to the API

We made no changes to the API.

V. Missing or Incomplete Elements of Our Code

There are no missing or incomplete elements in our code.

VI. Logistics

We spent approximately 25 man-hours on the project.

For selectivity estimation, in the special case of width<1, an int could map to multiple bins even though it would only be placed in one. In this case, the width would have to be scaled by an appropriate factor to account for the other, unused bins, and this is done by replacing width by width<1?1:width in the estimateSelectivity() computations. Previously, the lack of this compensation had resulted in an overestimation by a factor of 3 for computing the selectivity for equality with Min=0, Max=31; this was because width was computed as 32/100=0.32, when the width of a bin, e.g. for the number 4, should actually be 1.

Regarding the time of this assignment's submission, it has been submitted 1 day late; previously, we had 4 slip days available since we had not used any before.