Jason Tieu
304047667

Lab 3 Write Up

Exercise 1: Parser.java

Life of a query in SimpleDB (cont.):

Step 3: simpledb.Parser.handleQueryStatement()

- It creates a logical plan using parseQueryLogicalPlan(tId, s); tId is the Transaction ID passed into the method, and s is the Zquery. (Broken down in Step 4)
- After creating the logical plan, the Parser calls the method physicalPlan(TableStates.getStatsMap()) on the logical plan instance that was just created. The physical plan is returned as a DbIterator. (Broken down in Step 5)
- After these two are created, we can return the Query object representing this query

Step 4: simpledb.Parser.parseQueryLogicalPlan()

- First we create a new Logical plan
- Using the select clause (ZQuery q), a FROM clause is created from it (q.getFrom()).
- Next, we scan FROM clause and then parse the WHERE clause by using q.getWhere().
- processExpression determines what the ZExpression is (AND, OR, comparing constants, whether or not operands are nested queries).
- Next we look for group-by fields using q.getGroupBy(). In this implementation, at most one grouping field expression is supported. More than one is not supported.
- After getting the groupby field, we now look at the select list by using q.getSelect().
- We call Logical Plan's addProjectField(aggField, aggFun) to add a specified field/aggregate combo to the select list of the query.
- If there is an ORDER BY clause, we must sort the data accordingly. There could only be one element in the ORDER BY clause.
- Once all these steps are complete, we return the newly created Logical plan that has matched this query.

Step 5: simpledb.LogicalPlan.physicalPlan()

- We now must construct a physical plan from the logical plan, returned as a DbIterator
- First, we iterate through the tables from the WHERE clause and store each table
- Next we iterate through a filters vector, which contains the parameters of a filter in the WHERE clause of a query (LogicalFilterNode lf).
- Next we call the orderJoins() method from a newly created JoinOptimizer object, which computes a join on specified tables and returns a vector of LogicalJoinNodes (represents the state needed of a join of two tables in a LogicalQueryPlan) called joins.
- We then iterate through joins and call the instantiateJoin() method of the JoinOptimizer that returns the best iterator for computing a given logical join, provided with stats and left and right subplans.
- Then we go through the select list and determine the order of how to project the output fields.

Design Decisions

Due to issues with QueryTest, I have decided to use the solution to lab 2 in place of my own code.

- For selectivity estimation, I followed the basic guidelines of 2.2.3 Filter Selectivity.
- For join ordering, I followed the pseudocode guideline of 2.3 Join Ordering. I used the PlanCache class to store the best way to join a subset of the current joins being looked at. computeCostAndCardOfSubplan() gave me the cost and cardinality of a subplan which made it easier to find the best way to order the joins.

- My other designs stuck very close to what the documentation mentioned. I did not make any drastic changes.
- I was able to pass all the systemtests and tests

Time spent on lab: I worked on the lab on and off again for about two weeks. I found it challenging to implement orderJoins and kept running into issues when running the tests and systemtests.

5.1 Execute following query

0.01 (1% version of the IMDB database)

As shown by the 5.1 query plan below, the optimizer selected this particular plan with the smallest cardinality always on the left(outer). Selections are pushed on the right hand side. In real-world settings, this makes use of any available indexes and makes the cost alot less than it could be without pushing selects.

5.2 Excute another SQL query

0.01 (1% version of the IMDB database)

```
select d.fname, d.lname
from Actor a, Casts c, Movie_Director m, Director d, Movie mov, Genre g
where a.id=c.pid and c.mid=m.mid and m.did=d.id
and m.mid=mov.id and mov.id = g.mid
and mov.year>2000 and g.genre='Western';
```

The query plan is:

The 5.2 plan also shows that the lowest cardinality is on the right-hand side. The plan demonstrates a left-deep plan with the format of at most two relations being joined, and using its output to join with an inner relation on the right side of the branch. This plan shows a chain of joins with selects being pushed down for optimization whenever possible.

```
The query plan is:
                                  π(d.fname,d.lname),card:29729
                                  |
                  ⋈(a.id=c.pid),card:29729
  ________________________________|_____________________________________
  |                                                                     |
  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(Actor a)                     scan(Director d)  scan(Movie_Director m)  scan(Casts c)
```

5.1 Query Plan

```
The query plan is:
                                                π(d.fname,d.lname),card:29729
                                                |
                                  ⋈(g.mid=mov.id),card:29729
  ______________________________________________|_____________________________________________
  |                                                                                           |
  σ(g.genre=Western),card:1                                                                   ⋈(m.did=d.id),card:29729
  |                                                                                           |
  |                                                                         ____________________|_____________________
  |                                                                         |                                        |
  |                                                                         ⋈(m.mid=mov.id),card:29729               |
  |                                                          _______________|_____________                           |
  |                                                          |                           |                          |
  |                                                          ⋈(c.mid=m.mid),card:29729   σ(mov.year>2000),card:1236 |
  |                                             _____________|_____________              |                          |
  |                                             |                         |              |                          |
  |                                             ⋈(a.id=c.pid),card:29729   |              |                          |
  |                                  ___________|___________              |              |                          |
  |                                  |                     |              |              |                          |
scan(Genre g)                  scan(Actor a)  scan(Casts c)  scan(Movie_Director m)  scan(Movie mov)  scan(Director d)
```

5.2 Query Plan