CS 345 - Topics in Data Warehousing

Autumn Quarter, 2004

Assignment #3 – Due Tuesday, November 16

This programming assignment has three parts. In the first part, you will implement a greedy algorithm that selects a set of aggregate tables to accelerate query performance for a given database and workload. In the second part, you will write a program that creates the aggregate tables selected by the algorithm of Part 1. In the third part, you will implement an aggregate navigator that rewrites SQL queries to use the aggregate tables created in Part 2, rather than using the base fact and dimension tables.

This document describes the assignment at a fairly high level. More detailed information is available on the Assignments section of the course web page. **Additional information is available from http://cs345.stanford.edu/ in the "Assignments" section.**

Aggregate Table Selection In the first phase of the assignment, you will implement the greedy aggregate table selection algorithm that was described in class on November 2. The structure of the star schema for which the aggregates should be built will be described in a set of metadata tables (described below).

The running time of the greedy selection algorithm can be excessive when all possible combinations of grouping columns are considered. For this reason, the search space explored by the greedy algorithm will be somewhat restricted. An initial set of candidate column sets will be specified in the metadata tables. The aggregates that should be considered during the greedy search are the ones defined as follows: Join the fact to all its dimensions, group by a set of columns that is the union of zero or more candidate column sets, and sum over all measurement columns. In other words, if there are n candidate column sets, there are 2^n possible aggregates to consider.

For example, if the fact F has two dimensions D1 and D2, and there are three candidate column sets {D1.A}, {D1.B, D2.C}, and {D2.D, D2.E}, then there will be 8 possible aggregates, defined by the following 8 sets of grouping columns: $\emptyset$, {D1.A}, {D1.B, D2.C}, {D2.D, D2.E}, {D1.A, D1.B, D2.C}, {D1.A, D2.D, D2.E}, {D1.B, D2.C, D2.D, D2.E}, and {D1.A, D1.B, D2.C, D2.D, D2.E}.

The cost model to use when selecting aggregates is that the cost of a query is proportional to the number of rows in the fact aggregate table used to answer the query. The number of rows in a potential fact aggregate table can be computed without actually constructing the aggregate by performing a `COUNT DISTINCT` query. The query workload to use for cost evaluation is specified in the metadata tables, as is the maximum number of aggregate tables that should be created.

Your program for the first phase will not produce any output. Instead, it will populate the aggregate navigation metadata tables that will be used for the second and third phases of the assignment.

Aggregate Table Construction In the second phase of the assignment, you will write a program that constructs the fact and dimension aggregate tables recommended by the selection algorithm from Phase 1. Each fact aggregate table should have the same schema as the fact table, except that some of the dimension foreign keys will possibly be omitted. However, the foreign keys in the fact aggregate tables will generally point to dimension aggregate tables rather than to the ordinary dimension tables.

Your program for the second phase will not produce any output. Instead, it will create aggregate tables and fill them with the correct pre-computed summary data. The summary data can be computed by executing queries against the original fact and dimension tables.

Query Rewriting In the third phase of the assignment, you will write a program that performs aggregate navigation and re-writes SQL queries. Your program will read in a query from stdin, parse the query to determine which dimension columns are referenced, select the smallest fact aggregate table that can be used to answer the query, re-write the query to use the appropriate fact and dimension aggregates, and output the re-written SQL query to stdout.

To make the queries easy to parse, the SQL queries that are provided on stdin will have a very specific, highly constrained structure. More information about the query format is available form the course web site under the "Assignments" section. In addition, sample parsing code in C and Java is provided in the directory `/usr/class/cs345/HW3`. You are free to modify this code and incorporate it in your program.

Metadata Tables Information about the structure of the star schema and the query workload to use when selecting aggregates will be provided through a set of metadata tables. These tables can be created and populated using the SQL scripts in the directory `/usr/class/cs345/HW3`. More information about the structure of these tables can be found on the course web site under the "Assignments" section.

Constructing the Star Schema A small data warehouse with some test data will be provided for you to use to test your program. SQL scripts to copy the warehouse into your own Oracle area are provided in the directory `/usr/class/cs345/HW3`.

What To Submit Submissions for this assignment will be performed electronically. You should create a directory in your Stanford UNIX account to hold the files you create for this assignment. The directory should include (1) a README file describing which source code is relevant to which phases of the assignment, and (2) any code that you write, such as C or Java source files, SQL scripts, etc.

When you are ready to submit your assignment, "cd" to your assignment directory and then run:
`/usr/class/cs345/submit`
If you discover an error after submitting, you can re-submit by running the submit script with the `-replace` command-line argument.