CS562 MF Query Evaluator Final Project

Aaron Camacho and Walker Bove

Introduction

This presentation will describe the code that makes up our MF Query Evaluator, as well as provide a few sample queries to demonstrate how it works.

The Evaluator was written in Python, and generates Python code files that access the database and compute the results. Due to the simplicity of Python, the entire evaluator exists within a single Python file.

Purpose

While the SQL language can express any possible query that can be expressed, creating subgroups within the GROUP BY clause can become very wordy and inefficient, requiring the creation of multiple tables/views and joins (a notoriously expensive operation).

The MF Query Generator was designed to compute these queries without the view creation and expensive join costs normally associated with these types of queries.

Code Overview

Query 1

“Find for each customer the average sale in ‘NY’, the average sale in ‘CT’ and the average sale in ‘NJ’, if New York’s average is greater than the other two”

MF Input:

S: cust, avg_1_quant, avg_2_quant, avg_3_quant

N: 3

V: cust

F: avg_1_quant, avg_2_quant, avg_3_quant

Θ: record[“state”]==’NY’, record[“state”]==’NJ’, record[“state”]==’CT’

G: aggs.avg_1_quant[0]/aggs.avg_1_quant[1] > aggs.avg_2_quant[0]/aggs.avg_2_quant[1] and aggs.avg_1_quant[0]/aggs.avg_1_quant[1] > aggs.avg_3_quant[0]/aggs.avg_3_quant[1]

1 → NY

2 → NJ

3 → CT

Output and Equivalent SQL Query

Query 2

“Find for each customer the total number of sales for apples and the total number of sales for cherries if the customer has purchased more apples than cherries”

MF Input:

S: cust, sum_1_quant, sum_2_quant

N: 2

V: cust

F: sum_1_quant, sum_2_quant

Θ: record[“prod”]==’Apple’,
record[“prod”]==’Cherry’

G: aggs.sum_1_quant > aggs.sum_2_quant

1 → Apples

2 → Cherries

Output and Equivalent SQL Query

Query 3

"For each customer and product combination, find the minimum amount of product purchased in 2017, 2018, and 2019 where the minimum amount has decreased each year"

MF Input:

S: cust, prod, min_1_quant, min_2_quant, min_3_quant

N: 3

V: cust, prod

F: min_1_quant, min_2_quant, min_3_quant

Θ: record["year"]==2017, record["year"]==2018, record["year"]==2019

G: aggs.min_1_quant > aggs.min_2_quant and aggs.min_2_quant > aggs.min_3_quant

1 → 2017

2 → 2018

3 → 2019

Output and Equivalent SQL Query

Query 4

“For each customer and product, find the average sales quantities for each sales quarter, as well as the average and number of sales for the whole year”

MF Input:

S: cust, prod, avg_1_quant, avg_2_quant, avg_3_quant, avg_4_quant, avg_0_quant, count_0_quant

N: 4 (Note, gv 0 is considered implicit)

V: cust, prod

F: avg_0_quant, count_0_quant, avg_1_quant, avg_2_quant, avg_3_quant, avg_4_quant

Θ: record[“month”]>=1 and record[“month”]<=3, record[“month”]>=4 and record[“month”]<=6, record[“month”]>=7 and record[“month”]<=9, record[“month”]>=10 and record[“month”]<=12

G: True

0 → Whole Year

1 → Quarter 1

2 → Quarter 2

3 → Quarter 3

4 → Quarter 4

Output and Equivalent SQL Query

Things to Improve

- Support for EMF queries
- Error checking (presence of tables, valid aggregation, etc.)
- Minimal scanning detection (instead of all grouping variables being done one at a time)
- More advanced input processing (make the input rely less on existing Python syntax)