Programming Assignment #2 CAS CS 460

Query Optimization & Row-stores vs. Column-stores

Due: 12/2 11:59 pm on gradescope.

This programming assignment is for groups of two.

General

First make sure that you have PostgreSQL and MonetDB installed in your system. This file contains the instructions for the installation. Also, make sure that the database server runs in the background. You will need PostgreSQL for both task1 and task2. MonetDB will only be used for task2.

Task 1 (50%)

1.1 Introduction

In this task, you will carry out several exercises involving the optimization of relational queries using the PostgreSQL query optimizer and the visualization command EXPLAIN. You need to read parts of the PostgreSQL documentation to be able to complete this task. To be specific, you need to get familiar with the EXPLAIN, ANALYZE and the INFORMATION_SCHEMA Table commands of PostgreSQL. (links are provided in the resources section)

1.2 Setup

First, make sure that PostgreSQL runs in the background. Create a new database in PostgreSQL named 'task1database'. To do that, open terminal and type 'psql'. After that, type 'create database task1database'. Then, you will need to download this folder from here. Open this folder and run the setup.sh shell script. **Before running this shell script, make sure that you modify the loaddata.sql file, which is in the same folder, by changing the full path of the files**. This will create all databases and load the data for you. If you are not sure how to run a shell script, please check the resources section of this task.

For example, when you open the loaddata.sql you will see something like this:

```
COPY part FROM 'YOUR_FULL_PATH_HERE/part.csv' WITH (FORMAT csv, DELIMITER '|');
```

You will have to modify it to something like this:

```
COPY part FROM '/Users/dimitrisstaratzis/Desktop/pa2-task1/part.csv' WITH (FORMAT csv, DELIMITER '|');
```

Relation Schema:

We will use three tables in this experiment: part, supplier, partsupp, and lineitem.

part (p_partkey integer, p_name varchar(55), p_mfgr character(25), p_brand character (10), p_type varchar(25), p_size integer, p_containercharacter(10), p_retailprice numeri c(20,2), p_comment varchar(23), primary key (p_partkey));

supplier (s_suppkey integer, s_name char(25), s_address varchar(40), s_nationkey inte ger, s_phone character(15), s_acctbal numeric(20,2), s_commentvarchar(101), primary key (s_suppkey));

partsupp (ps_partkey integer, ps_suppkey integer, ps_availqty integer, ps_supplycost n umeric(20,2), ps_comment varchar(199), primary key(ps_partkey, ps_suppkey));

lineitem(l_orderkey integer, l_partkey integer, l_suppkey integer, l_linenumber integer, l_quantity numeric(20,2), l_extendedprice numeric(20,2), l_discountnumeric(3,2), l_tax numeric(3,2), l_returnflag character(1), l_linestatus character(1), l_shipdate date, l_com mitdate date, l_receiptdate date, l_shipinstructcharacter(25), l_shipmode character(10), l_comment varchar(44), primary key (l_orderkey, l_linenumber);

1.3 Exercises

When EXPLAIN is used with an explainable statement, PostgreSQL displays information from the optimizer about the statement execution plan. That is, PostgreSQL explains how it would process the statement, including information about how tables are joined and in which order. In general, use - EXPLAIN (ANALYZE true, COSTS true, FORMAT json) - to get the evaluation plan because it gives much more information about the plan. Use the actual execution of the query on terminal or profile information for query execution times.

1.3.1 Statistics of the tables

We will first examine the statistics for table lineitem. Answer the following questions.

1. How many records are there actually in “lineitem”? What is the estimated value by the query optimizer? How do you find these values (command or SQL)?
2. Is the value used by the query optimizer exact? If not, why?

1.3.2 Index on perfect match query

We will check how index affects query optimization and performance.

Examine the following query:

SELECT * FROM lineitem WHERE L_TAX = 0.07;

1. What is the estimated total cost of executing the best plan? What does the cost of a plan mean in MySQL?
2. What is the estimated result cardinality for this plan? How does the query optimizer obtain this value? Is it a reasonable one?
3. Which access method (access path) does the optimizer choose?

Create an index "ltax_idx" on the attribute "L_TAX".

4. Which access method does the optimizer consider to be the best now? Is the estimated result cardinality better now? Why?
5. Compare the two plans (without and with index). Explain briefly why access method in (4) is cheaper than the previous one.

1.3.3 Index on range select

Consider the following query:

SELECT * FROM lineitem WHERE L_QUANTITY < 45;

1. How many tuples does the query optimizer think will be returned? What is the estimated total cost?
2. What is the access method used?

Create an index "l_qty_idx" on the attribute "L_QUANTITY". Consider now the following query:

SELECT * FROM lineitem WHERE L_QUANTITY < 3;

3. What is the estimated total cost now? Is it correct? In what order would the tuples be returned by this plan?
4. Explain why one of the access methods is more expensive than the other.

1.3.4 Join algorithm

Consider the following query:

SELECT DISTINCT s_name

FROM supplier, partsupp

WHERE s_suppkey = ps_suppkey AND ps_availqty < 40;

Answer the follow questions:

1. Write down the best plan estimated by the optimizer (in plan tree form). What is the estimated total cost?
2. What is the join algorithm used in the plan? Explain how the system reads the two relations (what access path uses).
3. According to the optimizer, how many tuples will be retrieved from partsupp? How many from supplier? Are these estimations correct?
4. Can you add an index to improve the performance of the plan? Which index you will create and on which attribute? What is the new plan that is executed and what is its cost?
5. After you created the index, check the estimation of the tuples retrieved from partsupp. Is it correct? If yes, why?

1.4 Resources

Explain: https://www.postgresql.org/docs/9.4/using-explain.html

Analyze: https://www.postgresql.org/docs/9.1/sql-analyze.html

Information Schema: https://www.postgresql.org/docs/9.1/information-schema.html

Steps execute a script:

- Open the terminal. Go to the directory where you have your **script**.
- **Run** the **script** by typing: ./ScriptName.sh

Task 2 (50%)

2.1 Introduction

A common question in production is which system to use for a specific use-case. A good data engineer can provide such answers through experience, benchmarking, and intuition. The goal of this task is to start building these skills, starting with benchmarking.

Traditional DBMS architectures today follow two main approaches: a row-*major* and *column-major* approach. In this task you will study these approaches. We will use PostgreSQL as a row-major system and MonetDB as a column-major system. The goal of the project is to compare the performance of these two systems for a set of analytical queries taken from an industry-grade database systems benchmark.

2.2 Set-up

Create a new database in PostgreSQL named 'task2database_psql'. To do that, open terminal and type 'psql'. After that, type 'create database 'task2database_psql'. Do the same for MonetDB and create a database named 'task2database_monet'. Then you will need to load data in those two databases. To do that, download this folder from here and

first navigate to the **/dbgen** folder. In this folder run the make command. If you are not familiar with what make is or what it does, you can learn by navigating to the resources section of this part. This will generate the data, which you can examine if you like, in the /data folder. After that, navigate to **/monet-setup** and **/psql-setup**. Both these folders contain a file named **"tpch-load.sql".** Open each one of these files and modify the path.

For example, when you open the **psql-setup/tpch-load.sql** file you will see something like this:

```
COPY part FROM 'YOUR_FULL_PATH_HERE/part.csv' WITH (FORMAT csv, DELIMITER '|');
```

You will have to modify it to something like this:

```
COPY part FROM '/Users/dimitrisstaratzis/Desktop/pa2-task1/part.csv' WITH (FORMAT csv, DELIMITER '|');
```

The same applies for the **monte-setup/thcp-load.sql** file. Then, go back and run the setup.sh shell script. This will automatically load all data in both PostgreSQL and MonetDB. In the **/queries** folder there are some queries that you can choose for the exercises. See below.

2.3 Exercise

You will have to prepare a document where the two systems will be compared for 8 different queries from the TPC-H benchmark. You are free to select any 8 queries and present their performance (query latency). The reported performance should be accompanied with experimental setup, any tuning done to the system, and information with regards to standard deviation. Finally, the report should discuss which of the two systems is preferable for what type of queries based on the observed results and the intuition developed throughout the experimentation with the systems.

2.3 Resources

"Make" documentation
TPC-H Benchmark Spec File (go over Chapters 0, 1, and 2)

Submission instructions:

Please submit a PDF file in gradescope. **Make sure that you submit as a group! In this file include your full names and BU IDs.** Try to be as detailed and precise as possible while answering the questions!