PROJECT 2: UNDERSTANDING YOUR FRIENDLY QUERY OPTIMIZER

CX4031 DATABASE SYSTEM PRINCIPLES

TOTAL MARKS: 100

Due Date: Nov 20, 2020; 5 PM

A DBMS query optimizer executes a query execution plan (QEP) to process a query. As a database administrator, you would like to understand and visualize the performance of the query optimizer for processing queries. This is a typical task an administrator may perform in an organization to comprehend why certain queries or applications are slow. Fortunately, you do not have to write a software to do it. There is a tool called PICASSO (http://dsl.cds.iisc.ac.in/projects/PICASSO/) that can aid you to understand and visualize the performance of a query optimizer.

PROJECT DESCRIPTION

Download the PICASSO tool and install it on your database. Guideline for installing it detailed in the *Appendix*. Note that you may need to update the JDBC driver to make it work. Specifically, you will use the PICASSO tool to achieve the following two key goals of the project:

(a) **Generate and describe the plan, cost, and cardinality diagrams for a set of queries**. You should use the TPC-H benchmark data and queries described below. You are encouraged to read the software documentation of PICASSO (https://dsl.cds.iisc.ac.in/projects/PICASSO/picasso_download/doc/Picasso2Doc.pdf) as well related information in the PICASSO website to get familiarize with the concepts related to plan, cost, and cardinality estimation diagrams.

(b) **Design and implement an algorithm** that takes as input a query, retrieves its query execution plan, and returns as output an **explanation** (i.e., reason) that describes the reason why the underlying RDBMS selected this plan among many others. You should exploit the PICASSO tool to generate the explanation. Note that the structure of explanation is open-ended. It can be in the form of natural language or in a more structured form. You are encouraged to develop a visual interface that facilitates user-friendly interaction with your software.
Your goal is to ensure generality of the solution (i.e., it can handle a wide variety of queries) and the explanation should be **precise and accurate**. *Hint: You need to transform the input query to a PICASSO query template in order to generate*

explanation. Check the guideline in the Appendix to see which part of the PICASSO code you need to understand.

You should use **Python or Java** as the host language on **Windows** platform for your project. The DBMS allowed in this project is **PostgreSQL.** The dataset and queries you should use for this project is **TPC-H** (see *Appendix*). You are free to use any off-the-shelf toolkits for your project.

For students using **Mac** platform, you can **install Windows on your Mac** by following instructions in https://support.apple.com/en-sg/HT201468.

Note that several parts of the project are left open-ended (e.g., what will be the content of the explanation? how the GUI should look like? What are the functionalities we should support?) deliberately so that the project does not curb a group's creative endeavors. You are free to make realistic assumptions to achieve these tasks.

SUBMISSION REQUIREMENTS

You submission should include the followings:

- A **report** containing (a) the modified TPC-H benchmark queries to generate the plan, cost, and cardinality diagrams; (b) hard copy of screenshots of these diagrams; (c) description and analysis of the performance of the query processor using these diagrams; and (d) detailed description of the algorithm (with examples) used to address *Part B*.
- **Source code** of *Part B*. Note that we will execute the software to check its correctness using **different** query sets to check for the generality of the solution.
- You must submit a document containing instructions to run your software (*Part B*) successfully. You will not receive any credit if your software fails to execute based on your instructions.

Note: Late submission will be penalized.

Appendix

I. Creating TPC-H database in PostgreSQL

Follow the following steps to generate the TPC-H data:

1) Go to http://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp and download TPC-H Tools v2.18.0.zip. Note that the version may defer as the tool may have been updated by the developer.
2) Unzip the package. You will find a folder "dbgen" in it.
3) To generate an instance of the TPC-H database:
    - Open up tpch.vcproj using visual studio software.
    - Build the tpch project. When the build is successful, a command prompt will appear with "TPC-H Population Generator <Version 2.17.3>" and several *.tbl files will be generated. You should expect the following .tbl files: customer.tbl, lineitem.tbl, nation.tbl, orders.tbl, part.tbl, partsupp.tbl, region.tbl, supplier.tbl
    - Save these .tbl files as .csv files
    - These .csv files contain an extra " | " character at the end of each line. These " | " characters are incompatible with the format that PostgreSQL is expecting. Write a small piece of code to remove the last " | " character in each line. Now you are ready to load the .csv files into PostgreSQL
    - Open up PostgreSQL. Add a new database "TPC-H".
    - Create new tables for "customer", "lineitem", "nation", "orders", "part", "partsupp", "region" and "supplier"
    - Import the relevant .csv into each table. Note that pgAdmin4 for PostgreSQL (windows version) allows you to perform import easily. You can select to view the first 100 rows to check if the import has been done correctly.
      If encountered error (e.g., ERROR: extra data after last expected column) while importing, create columns of each table first before importing. Note that the types of each column has to be set appropriately. You may use the SQL commands in Appendix II to create the tables.

II. SQL commands for creating TPC-H data tables

Region table

```
CREATE TABLE public.region
(
    r_regionkey integer NOT NULL,
    r_name character(25) COLLATE pg_catalog."default" NOT NULL,
    r_comment character varying(152) COLLATE pg_catalog."default",
    CONSTRAINT region_pkey PRIMARY KEY (r_regionkey)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.region
    OWNER to postgres;
```

1) Nation table

```
CREATE TABLE public.nation
(
    n_nationkey integer NOT NULL,
    n_name character(25) COLLATE pg_catalog."default" NOT NULL,
    n_regionkey integer NOT NULL,
    n_comment character varying(152) COLLATE pg_catalog."default",
    CONSTRAINT nation_pkey PRIMARY KEY (n_nationkey),
    CONSTRAINT fk_nation FOREIGN KEY (n_regionkey)
        REFERENCES public.region (r_regionkey) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.nation
    OWNER to postgres;
```

2) Part table

```
CREATE TABLE public.part
(
    p_partkey integer NOT NULL,
    p_name character varying(55) COLLATE pg_catalog."default" NOT NULL,
    p_mfgr character(25) COLLATE pg_catalog."default" NOT NULL,
    p_brand character(10) COLLATE pg_catalog."default" NOT NULL,
    p_type character varying(25) COLLATE pg_catalog."default" NOT NULL,
    p_size integer NOT NULL,
    p_container character(10) COLLATE pg_catalog."default" NOT NULL,
    p_retailprice numeric(15,2) NOT NULL,
    p_comment character varying(23) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT part_pkey PRIMARY KEY (p_partkey)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.part
    OWNER to postgres;
```

3) Supplier table

```
CREATE TABLE public.supplier
(
    s_suppkey integer NOT NULL,
    s_name character(25) COLLATE pg_catalog."default" NOT NULL,
    s_address character varying(40) COLLATE pg_catalog."default" NOT NULL,
    s_nationkey integer NOT NULL,
    s_phone character(15) COLLATE pg_catalog."default" NOT NULL,
    s_acctbal numeric(15,2) NOT NULL,
    s_comment character varying(101) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT supplier_pkey PRIMARY KEY (s_suppkey),
    CONSTRAINT fk_supplier FOREIGN KEY (s_nationkey)
        REFERENCES public.nation (n_nationkey) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.supplier
    OWNER to postgres;
```

4) Partsupp table

```
CREATE TABLE public.partsupp
(
    ps_partkey integer NOT NULL,
    ps_suppkey integer NOT NULL,
    ps_availqty integer NOT NULL,
    ps_supplycost numeric(15,2) NOT NULL,
    ps_comment character varying(199) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT partsupp_pkey PRIMARY KEY (ps_partkey, ps_suppkey),
    CONSTRAINT fk_ps_suppkey_partkey FOREIGN KEY (ps_partkey)
        REFERENCES public.part (p_partkey) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_ps_suppkey_suppkey FOREIGN KEY (ps_suppkey)
        REFERENCES public.supplier (s_suppkey) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.partsupp
    OWNER to postgres;
```

5) Customer table

```
CREATE TABLE public.customer
(
    c_custkey integer NOT NULL,
    c_name character varying(25) COLLATE pg_catalog."default" NOT NULL,
    c_address character varying(40) COLLATE pg_catalog."default" NOT NULL,
    c_nationkey integer NOT NULL,
    c_phone character(15) COLLATE pg_catalog."default" NOT NULL,
    c_acctbal numeric(15,2) NOT NULL,
    c_mktsegment character(10) COLLATE pg_catalog."default" NOT NULL,
    c_comment character varying(117) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT customer_pkey PRIMARY KEY (c_custkey),
    CONSTRAINT fk_customer FOREIGN KEY (c_nationkey)
        REFERENCES public.nation (n_nationkey) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.customer
    OWNER to postgres;
```

6) Orders table

```
CREATE TABLE public.orders
(
    o_orderkey integer NOT NULL,
    o_custkey integer NOT NULL,
    o_orderstatus character(1) COLLATE pg_catalog."default" NOT NULL,
    o_totalprice numeric(15,2) NOT NULL,
    o_orderdate date NOT NULL,
    o_orderpriority character(15) COLLATE pg_catalog."default" NOT NULL,
    o_clerk character(15) COLLATE pg_catalog."default" NOT NULL,
    o_shippriority integer NOT NULL,
    o_comment character varying(79) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT orders_pkey PRIMARY KEY (o_orderkey),
    CONSTRAINT fk_orders FOREIGN KEY (o_custkey)
        REFERENCES public.customer (c_custkey) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.orders
    OWNER to postgres;
```

7) Lineitem table

```
CREATE TABLE public.lineitem
(
    l_orderkey integer NOT NULL,
    l_partkey integer NOT NULL,
    l_suppkey integer NOT NULL,
    l_linenumber integer NOT NULL,
    l_quantity numeric(15,2) NOT NULL,
    l_extendedprice numeric(15,2) NOT NULL,
    l_discount numeric(15,2) NOT NULL,
    l_tax numeric(15,2) NOT NULL,
    l_returnflag character(1) COLLATE pg_catalog."default" NOT NULL,
    l_linestatus character(1) COLLATE pg_catalog."default" NOT NULL,
    l_shipdate date NOT NULL,
    l_commitdate date NOT NULL,
    l_receiptdate date NOT NULL,
    l_shipinstruct character(25) COLLATE pg_catalog."default" NOT NULL,
    l_shipmode character(10) COLLATE pg_catalog."default" NOT NULL,
    l_comment character varying(44) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT lineitem_pkey PRIMARY KEY (l_orderkey, l_partkey, l_suppkey, l_linenumber),
    CONSTRAINT fk_lineitem_orderkey FOREIGN KEY (l_orderkey)
        REFERENCES public.orders (o_orderkey) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_lineitem_partkey FOREIGN KEY (l_partkey)
        REFERENCES public.part (p_partkey) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT fk_lineitem_suppkey FOREIGN KEY (l_suppkey)
        REFERENCES public.supplier (s_suppkey) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.lineitem
    OWNER to postgres;
```

III. Installing and Using PICASSO

1) Download picasso2.1.zip from
https://dsl.cds.iisc.ac.in/projects/PICASSO/picasso_download/license.htm

Follow installation instruction in:
https://dsl.cds.iisc.ac.in/projects/PICASSO/picasso_download/doc/Installation/installation.htm

Note:
Install Java 3D 1.5.1 (https://www.oracle.com/java/technologies/java-archive-downloads-java-client-downloads.html#java3d-1.5.1-oth-JPR).
Upon installation, copy the three .jar (j3dcore.jar, j3dutils.jar and vecmath.jar) and j3dcore-ogl.dll (for Windows) to ..\picasso2.1\picasso2.1\Libraries.

2) Guide on how to use the PICASSO GUI is found here:
https://dsl.cds.iisc.ac.in/projects/PICASSO/picasso_download/doc/Usage/controls.htm

Note: For Part (b) of the project, you would need to explore the code of PICASSO. A good place to start is PicassoDiagram.java inside the PicassoServer folder. The .java provides the logic of generating the diagrams used in PICASSO GUI.