PROJECT 2: CAN QUERY EXECUTION PLANS HELP YOU TO GENERATE BETTER SQL?

SC3020 DATABASE SYSTEM PRINCIPLES

TOTAL MARKS: 100

Due Date: April 20, 2025; 11:59 PM

Real-world users may write SQL queries to search relational databases for different tasks. The RDBMS query optimizer will execute a query execution plan (QEP) to process each query, which is chosen from a large number of *alternative query plans* (AQP). SQL, however, is not an easy language to learn or use. Even for expert users, it is challenging to read, write, and work with. This is because the logic of the query is not expressed sequentially (e.g., the FROM (INPUT) clause comes after SELECT clause (output)).

PROJECT GOAL

The goal of this project is to exploit the QEP of a given SQL query to create an equivalent variant of the SQL using *pipe-syntax* to improve understanding of the query. We illustrate this goal with an example. Consider the SQL query (from the **TPC-H** dataset) in Figure 1.

```
SELECT c_count, count(*) AS custdist
FROM
     (SELECT c_custkey, count(o_orderkey)
      FROM customer LEFT OUTER JOIN orders
      ON c_custkey = o_custkey
      AND o_comment not like '%unusual%packages%'
      GROUP BY c_custkey
     ) as c_orders (c_custkey, c_count)
GROUP BY c_count
ORDER BY custdist DESC, c_count DESC;
```

Figure 1

It is difficult to understand the query as one needs to read it inside-out starting from the subquery. That is, the logic of the query does not follow top-down, sequential order which makes SQL hard to read and understand. The goal is to generate an equivalent logic of the query by exploiting the QEP of the query. Figure 2 shows the visual representation of the QEP of this query.

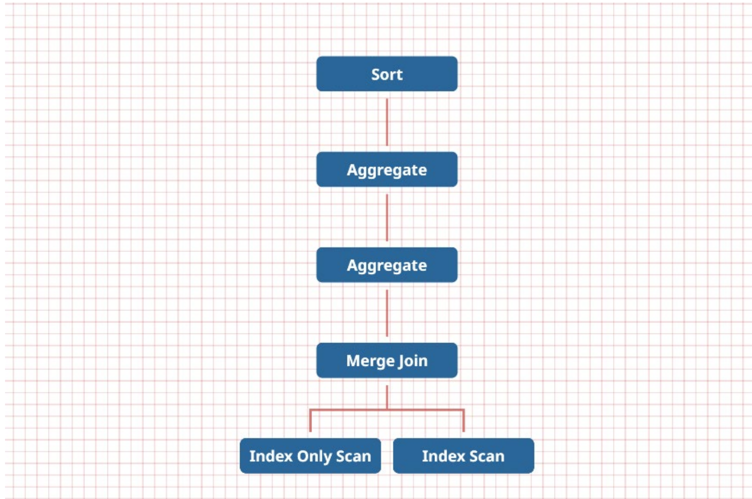

Figure 2

Observe that the QEP essentially explains the order in which various operations in the query are executed. Hence, the goal is to generate an equivalent logic as shown in Figure 3 by exploiting the SQL query and its QEP. Observe that the flow is top-down in sequential order and follows the QEP. The first two lines essentially maps to the join operation in Figure 2. The third and fourth lines map to the two aggregate operations in the QEP. The last line maps to the root note of the QEP. Also observe that these operations are separated by the pipe syntax (|>).

```
FROM customer
|> LEFT OUTER JOIN orders ON c_custkey = o_custkey AND o_comment NOT LIKE
'%unusual%packages%'
|> AGGREGATE COUNT(o_orderkey) c_count GROUP BY c_custkey
|> AGGREGATE COUNT(*) AS custdist GROUP BY c_count
|> ORDER BY custdist DESC, c_count DESC;
```

Figure 3

The goal of this project is to <u>automatically</u> generate the equivalent pipe-syntax format the input query by exploiting the QEP.

<u>Real-world implication of the project.</u> The benefit of the pipe-syntax version of the SQL query has real-world implications in the future. Recently, Google has extended SQL to this pipe-syntax version called *GoogleSQL* [1]. It is indeed a distinct possibility that this may become prevalent in the industry in the future. Hence, an auxiliary goal of this project is to put you at the leading edge of industrial development w.r.t. SQL.

<u>SPECIFIC GOALS</u>

(a) **Design and implement an efficient algorithm** that takes as input an SQL query, retrieves its query execution plan, and returns as output the pipe-syntax version of the query by exploiting the QEP. Note that the structure of output query is open-ended and you may exploit the ideas in [1]. In addition, you should annotate each component of the output query with the estimated cost as retrieved using the QEP. For example, the four components in Figure 3 should be annotated with the estimated cost. This will give an end user an idea of the estimated cost of different query components.

Your goal is to ensure generality of the solution (i.e., it can handle a wide variety of queries and should be independent of the underlying database schema). Hence, your software should work on any database schema and for a wide variety of SQL queries on it. In other words, you should not be hardcoding SQL queries or schema-specific information.

(b) **Design and implement a user-friendly graphical user interface (GUI)** to visualize your results. You can imagine that through it you can choose the database schema (TPC-H in this case), specify the query in the *Query panel* and upon execution of your algorithm the *Result panel* is updated with the pipe-syntax version of the query. You may also selectively visualize the corresponding QEP in another panel.

You should use **Python** as the host language on **Windows** platform for your project. Note that **Windows** is the official operating system in CCDS and all labs have it installed in most machines. The DBMS allowed in this project is **PostgreSQL.** The example dataset and queries you should use for this project is either **TPC-H (Appendix)** or **IMDB (https://developer.imdb.com/non-commercial-datasets/)**. An example of the relational schema of IMDB dataset is available at https://arxiv.org/pdf/2012.14800 (Fig 1). You are free to use any off-the-shelf toolkits for your project.

Note that several parts of the project are left deliberately open-ended (e.g., what will be the syntax of the pipe-syntax query? how the GUI should look like? What are the functionalities we should support?) 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:

- You should submit **four** program files: *interface.py*, *pipesyntax.py*, *preprocessing.py*, and *project.py*. The file *interface.py* contains the code for the GUI. The *pipesyntax.py* contains code for generating the pipe-syntax version of the query. The *preprocessing.py* file contains any code for reading inputs and any preprocessing necessary to make your algorithm work. Lastly, the *project.py* is the main file that invokes all the necessary procedures from these three files.

Note that we shall be running the project.py file (either from command prompt or using the Pycharm IDE) to execute the software. Make sure your code follows good coding practice: sufficient comments, proper variable/function naming, etc. We will execute the software to check its correctness using **different** query sets and dataset to check for the generality of the solution. We will also check quality of algorithm design w.r.t processing of the query plans.

- **Softcopy report** containing details of the software including formal descriptions of the key algorithms with examples. You should also discuss limitations of the software (if any).
- **Peer assessment report** from each member of the team. Each individual member of a team needs to assess contributions of the group members. Details of peer assessment form will be provided closer to the submission date.
- If you are using any **Large Language Model** (e.g., ChatGPT) for your project then you must acknowledge its usage appropriately and clearly specify where you have used it (in the report). Failure to do so may result in an F grade for the project.
- You must submit a document containing instructions to run your software successfully. You will not receive any credit if your software fails to execute based on your instructions.
- All submissions will be through NTU Learn. Details of submissions will be released nearer to the submission date.

Note: Late submission will be penalized (10 marks per day). No submission is allowed if you are late by more than 3 days.

GUIDELINES FOR TIME MANAGEMENT

The workload of the project is designed for a team of 4-5 students putting reasonable effort for a month. Hence, note the following guidelines.

- Start your project **early**. Delaying the start will only make it challenging to finish it on time.
- The project is a team project and not an individual effort. All are expected to contribute to the project. Distribute the workload equally so that a member can finish the work within 3-4 weeks.
- Seek clarification to doubts and queries whenever you need. You are advised to clarify any misunderstanding early so that you do not spend time on doing work that is not relevant to the project.

REFERENCE

[1] GoogleSQL: https://research.google/pubs/sql-has-problems-we-can-fix-them-pipe-syntax-in-sql/

Appendix

I. Creating TPC-H database in PostgreSQL

Follow the following steps to generate the TPC-H data (this step may differ slightly due to different versions of TPC-H). So you should use this as just a general guideline. Please modify the steps as deemed necessary.

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.

Alternatively, you can also refer to various online sources for additional help on creating the TPC-H database.

II. SQL commands for creating TPC-H data tables

Region table

```sql
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

```sql
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

```sql
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

```sql
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

```sql
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;
```