

PROJECT 2: WHAT-IF ANALYSIS OF QUERY PLANS

CX4031/SC3020 DATABASE SYSTEM PRINCIPLES

TOTAL MARKS: 100

Due Date: Nov 17, 2024; 11:59 PM

BACKGROUND

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). Typically, the plan selected as the QEP is estimated to have least/lower cost than other AQP. One can retrieve and view these plans in visual (tree-structured view) or text format (e.g., json, XML) in an off-the-shelf DBMS software (e.g., PostgreSQL). In particular, PostgreSQL allows one to retrieve AQPs containing specific operators using the *planner method configuration* (<https://www.postgresql.org/docs/current/runtime-config-query.html>).

OBJECTIVES

The broad aims of the project are as follows:

- Retrieve and visualize the QEP of a given SQL query.
- Support **what-if queries** on the QEP by enabling interactive modification of the physical operators and join order in the visual tree view of the QEP to generate an AQP.
- Retrieve the estimated cost of the AQP and compare its cost with the QEP.

PROJECT DESCRIPTION

The goal of the project can be described with an example. Consider the SQL query in Figure 1. The simplified view of the QEP for execution of this query in PostgreSQL is given on the right. Observe that as a user I may want to know what will be the impact on the cost if I change the hash join to merge join? Or what will be the cost when I replace the sequential scan with an index scan on one of the tables? The project aims to build a software to enable such **what-if** questions on QEPs. To this end, the project should address the following problems:

- Enable manipulation and editing of the visual tree view of the QEP to specify these what-if questions.
- Generate the corresponding SQL query based on the what-if question by exploiting the planner method configuration of PostgreSQL.
- Retrieve the AQP associated with the modified SQL and compare its cost with the original QEP.

```
select *
from   customer C,
orders O
where  C.c_custkey =
O.o_custkey
```

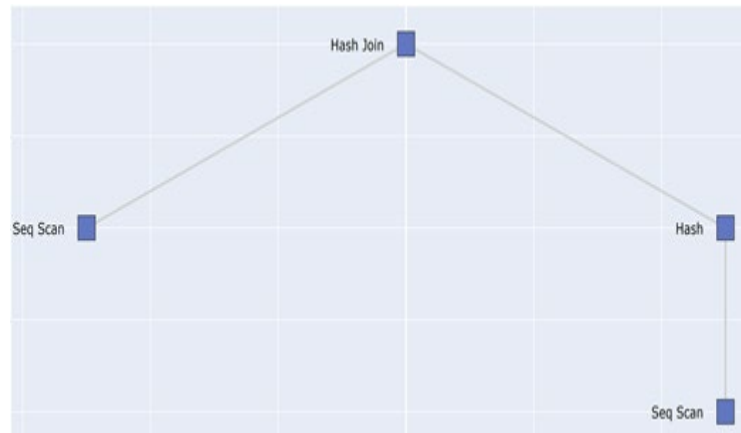


Figure 1

In order to realize the above goals, you are required to address the followings:

- (a) **Design and implement an efficient algorithm** that takes as input the original SQL query, the modified visual QEP with what-if questions, and returns as output a modified SQL query and its corresponding QEP (which is the AQP) that describes the execution of various components of the query based on the what-if questions. Additionally, the cost of the AQP needs to be compared with that of the original QEP to shed insights into the impact of the changes of the physical operators/join order on the estimated cost.

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 support your what-if analysis software. You can imagine that through it you can choose the database schema (TPC-H in this case), specify the SQL query in the *Query panel*, visualize the QEP on the *QEP panel* which you can interactively edit for posing what-if questions. Your GUI should also have a panel to visualize the

modified SQL generated based on the what-if question, and another panel to view the corresponding AQP along with cost comparison with the original QEP.

You should use **Python** as the host language on **Windows** platform 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>. The DBMS allowed in this project is **PostgreSQL**. The example 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 especially in implementing your GUI.

Note that several parts of the project are left deliberately open-ended (e.g., what will be the content of the annotations? 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

Your submission should include the followings:

- You should submit **four** program files: *interface.py*, *whatif.py*, *preprocessing.py*, and *project.py*. The file *interface.py* contains the code for the GUI (you may use any other GUI development toolkit as long as it is compatible with Python). The *whatif.py* contains code for generating the modified SQL and corresponding AQP. 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 and what-if questions.
- **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.

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 <https://docs.verdictdb.org/tutorial/tpch/> for additional help on creating the TPC-H database

II. SQL commands for creating TPC-H data tables

Region table

```
5 CREATE TABLE public.region
6 (
7     r_regionkey integer NOT NULL,
8     r_name character(25) COLLATE pg_catalog."default" NOT NULL,
9     r_comment character varying(152) COLLATE pg_catalog."default",
10    CONSTRAINT region_pkey PRIMARY KEY (r_regionkey)
11 )
12 WITH (
13     OIDS = FALSE
14 )
15 TABLESPACE pg_default;
16
17 ALTER TABLE public.region
18     OWNER to postgres;
```

1) Nation table

```
5 CREATE TABLE public.nation
6 (
7     n_nationkey integer NOT NULL,
8     n_name character(25) COLLATE pg_catalog."default" NOT NULL,
9     n_regionkey integer NOT NULL,
10    n_comment character varying(152) COLLATE pg_catalog."default",
11    CONSTRAINT nation_pkey PRIMARY KEY (n_nationkey),
12    CONSTRAINT fk_nation FOREIGN KEY (n_regionkey)
13        REFERENCES public.region (r_regionkey) MATCH SIMPLE
14        ON UPDATE NO ACTION
15        ON DELETE NO ACTION
16 )
17 WITH (
18     OIDS = FALSE
19 )
20 TABLESPACE pg_default;
21
22 ALTER TABLE public.nation
23     OWNER to postgres;
```

2) Part table

```
5 CREATE TABLE public.part
6 (
7     p_partkey integer NOT NULL,
8     p_name character varying(55) COLLATE pg_catalog."default" NOT NULL,
9     p_mfgr character(25) COLLATE pg_catalog."default" NOT NULL,
10    p_brand character(10) COLLATE pg_catalog."default" NOT NULL,
11    p_type character varying(25) COLLATE pg_catalog."default" NOT NULL,
12    p_size integer NOT NULL,
13    p_container character(10) COLLATE pg_catalog."default" NOT NULL,
14    p_retailprice numeric(15,2) NOT NULL,
15    p_comment character varying(23) COLLATE pg_catalog."default" NOT NULL,
16    CONSTRAINT part_pkey PRIMARY KEY (p_partkey)
17 )
18 WITH (
19     OIDS = FALSE
20 )
21 TABLESPACE pg_default;
22
23 ALTER TABLE public.part
24     OWNER to postgres;
```

3) Supplier table

```
5 CREATE TABLE public.supplier
6 (
7     s_suppkey integer NOT NULL,
8     s_name character(25) COLLATE pg_catalog."default" NOT NULL,
9     s_address character varying(40) COLLATE pg_catalog."default" NOT NULL,
10    s_nationkey integer NOT NULL,
11    s_phone character(15) COLLATE pg_catalog."default" NOT NULL,
12    s_acctbal numeric(15,2) NOT NULL,
13    s_comment character varying(101) COLLATE pg_catalog."default" NOT NULL,
14    CONSTRAINT supplier_pkey PRIMARY KEY (s_suppkey),
15    CONSTRAINT fk_supplier FOREIGN KEY (s_nationkey)
16        REFERENCES public.nation (n_nationkey) MATCH SIMPLE
17        ON UPDATE NO ACTION
18        ON DELETE NO ACTION
19 )
20 WITH (
21     OIDS = FALSE
22 )
23 TABLESPACE pg_default;
24
25 ALTER TABLE public.supplier
26     OWNER to postgres;
```

4) Partsupp table

```
5 CREATE TABLE public.partsupp
6 (
7     ps_partkey integer NOT NULL,
8     ps_suppkey integer NOT NULL,
9     ps_availqty integer NOT NULL,
10    ps_supplycost numeric(15,2) NOT NULL,
11    ps_comment character varying(199) COLLATE pg_catalog."default" NOT NULL,
12    CONSTRAINT partsupp_pkey PRIMARY KEY (ps_partkey, ps_suppkey),
13    CONSTRAINT fk_ps_suppkey_partkey FOREIGN KEY (ps_partkey)
14        REFERENCES public.part (p_partkey) MATCH SIMPLE
15        ON UPDATE NO ACTION
16        ON DELETE NO ACTION,
17    CONSTRAINT fk_ps_suppkey_suppkey FOREIGN KEY (ps_suppkey)
18        REFERENCES public.supplier (s_suppkey) MATCH SIMPLE
19        ON UPDATE NO ACTION
20        ON DELETE NO ACTION
21 )
22 WITH (
23     OIDS = FALSE
24 )
25 TABLESPACE pg_default;
26
27 ALTER TABLE public.partsupp
28     OWNER to postgres;
```

5) Customer table

```
5 CREATE TABLE public.customer
6 (
7     c_custkey integer NOT NULL,
8     c_name character varying(25) COLLATE pg_catalog."default" NOT NULL,
9     c_address character varying(40) COLLATE pg_catalog."default" NOT NULL,
10    c_nationkey integer NOT NULL,
11    c_phone character(15) COLLATE pg_catalog."default" NOT NULL,
12    c_acctbal numeric(15,2) NOT NULL,
13    c_mktsegment character(10) COLLATE pg_catalog."default" NOT NULL,
14    c_comment character varying(117) COLLATE pg_catalog."default" NOT NULL,
15    CONSTRAINT customer_pkey PRIMARY KEY (c_custkey),
16    CONSTRAINT fk_customer FOREIGN KEY (c_nationkey)
17        REFERENCES public.nation (n_nationkey) MATCH SIMPLE
18        ON UPDATE NO ACTION
19        ON DELETE NO ACTION
20 )
21 WITH (
22     OIDS = FALSE
23 )
24 TABLESPACE pg_default;
25
26 ALTER TABLE public.customer
27     OWNER to postgres;
```


6) Orders table

```
5 CREATE TABLE public.orders
6 (
7     o_orderkey integer NOT NULL,
8     o_custkey integer NOT NULL,
9     o_orderstatus character(1) COLLATE pg_catalog."default" NOT NULL,
10    o_totalprice numeric(15,2) NOT NULL,
11    o_orderdate date NOT NULL,
12    o_orderpriority character(15) COLLATE pg_catalog."default" NOT NULL,
13    o_clerk character(15) COLLATE pg_catalog."default" NOT NULL,
14    o_shippriority integer NOT NULL,
15    o_comment character varying(79) COLLATE pg_catalog."default" NOT NULL,
16    CONSTRAINT orders_pkey PRIMARY KEY (o_orderkey),
17    CONSTRAINT fk_orders FOREIGN KEY (o_custkey)
18        REFERENCES public.customer (c_custkey) MATCH SIMPLE
19        ON UPDATE NO ACTION
20        ON DELETE NO ACTION
21 )
22 WITH (
23     OIDS = FALSE
24 )
25 TABLESPACE pg_default;
26
27 ALTER TABLE public.orders
28     OWNER to postgres;
```

7) Lineitem table

```
5 CREATE TABLE public.lineitem
6 (
7     l_orderkey integer NOT NULL,
8     l_partkey integer NOT NULL,
9     l_suppkey integer NOT NULL,
10    l_linenumber integer NOT NULL,
11    l_quantity numeric(15,2) NOT NULL,
12    l_extendedprice numeric(15,2) NOT NULL,
13    l_discount numeric(15,2) NOT NULL,
14    l_tax numeric(15,2) NOT NULL,
15    l_returnflag character(1) COLLATE pg_catalog."default" NOT NULL,
16    l_linestatus character(1) COLLATE pg_catalog."default" NOT NULL,
17    l_shipdate date NOT NULL,
18    l_commitdate date NOT NULL,
19    l_receiptdate date NOT NULL,
20    l_shipinstruct character(25) COLLATE pg_catalog."default" NOT NULL,
21    l_shipmode character(10) COLLATE pg_catalog."default" NOT NULL,
22    l_comment character varying(44) COLLATE pg_catalog."default" NOT NULL,
23    CONSTRAINT lineitem_pkey PRIMARY KEY (l_orderkey, l_partkey, l_suppkey, l_linenumber),
24    CONSTRAINT fk_lineitem_orderkey FOREIGN KEY (l_orderkey)
25        REFERENCES public.orders (o_orderkey) MATCH SIMPLE
26        ON UPDATE NO ACTION
27        ON DELETE NO ACTION,
28    CONSTRAINT fk_lineitem_partkey FOREIGN KEY (l_partkey)
29        REFERENCES public.part (p_partkey) MATCH SIMPLE
30        ON UPDATE NO ACTION
31        ON DELETE NO ACTION,
32    CONSTRAINT fk_lineitem_suppkey FOREIGN KEY (l_suppkey)
33        REFERENCES public.supplier (s_suppkey) MATCH SIMPLE
34        ON UPDATE NO ACTION
35        ON DELETE NO ACTION
36 )
37 WITH (
38     OIDS = FALSE
39 )
40 TABLESPACE pg_default;
41
42 ALTER TABLE public.lineitem
43     OWNER to postgres;
```