CZ4031 - Database System Principles

Semester 1 Academic Year 21/22

Project 2 Report

Name	Matric Number	Email Address
Koh Tzi Yong	U1920076C	tkoh013@e.ntu.edu.sg
Pong Zhi Jun Jeremy	U1922951J	pong0012@e.ntu.edu.sg
Neo Qi Xiang	U1921697H	qneo001@e.ntu.edu.sg
Saw William Joseph	U1921246F	SAWW0002@e.ntu.edu.sg

Table of Contents

Introduction

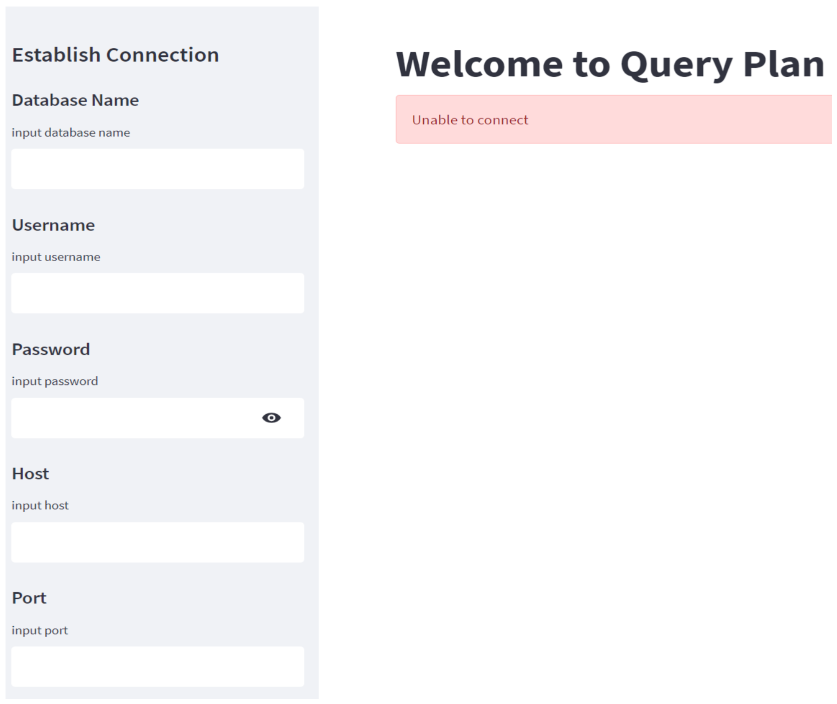
The objective of the query execution plan project is to allow users to visualise and understand how queries are being processed in SQL servers. For this project, postgresql server is used and the interface is built on a python module called streamlit.

The example queries run on this project are based on the TPC-H database. Feel free to connect to your own database to try out the QEP GUI. The instructions to load and run the interface are found in the other report.

GUI Overview

The streamlite[1] package is used to develop our GUI. Streamlite is an intuitive and powerful tool for data scientists to visualise their data projects.

This section shows our project interface.



Establish Connection

Database Name

input database name

Username

input username

Password

input password

Host

input host

Port

input port

Welcome to Query Plan

Unable to connect

Figure 1: Initial Web Page

[1] https://streamlit.io/

Users are to insert Database Name, Username of SQL server, Password, Host, Port. By default host and port is set to be localhost and 5432 respectively. Once the authentication is successful the GUI will update for us to execute our query.

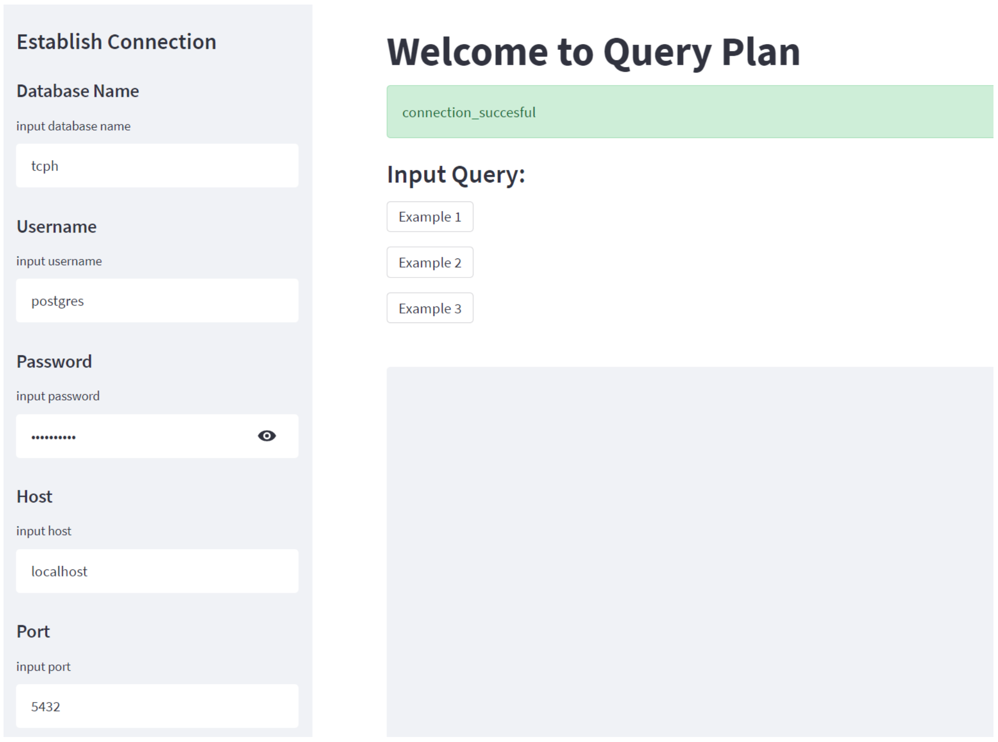
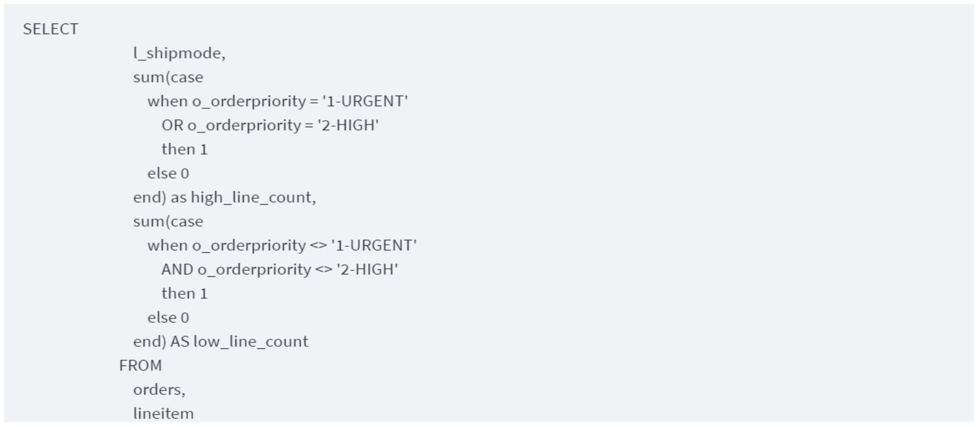


Figure 2: After successful authentication

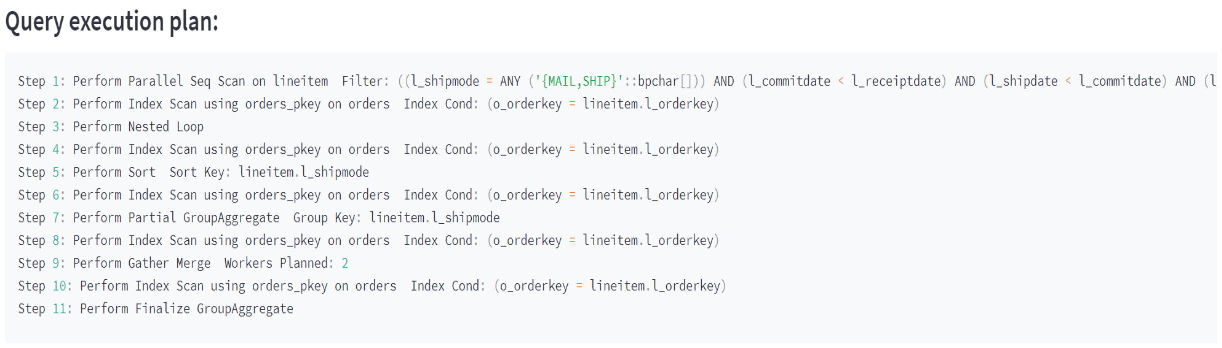
If the authentication parameters are set correctly, a green “connection successful” label will appear. Users are able to try out the example queries provided or input their own queries in the grey box below.



```
SELECT
        l_shipmode,
        sum(case
          when o_orderpriority = '1-URGENT'
            OR o_orderpriority = '2-HIGH'
            then 1
          else 0
        end) as high_line_count,
        sum(case
          when o_orderpriority <> '1-URGENT'
            AND o_orderpriority <> '2-HIGH'
            then 1
          else 0
        end) AS low_line_count
      FROM
        orders,
        lineitem
```

Figure 3: Example Query

Example:
If we select example 3 by clicking on “example 3”. Example 3 query will appear in the text box as shown above. The text box can be manipulated, you can try out your own queries too.

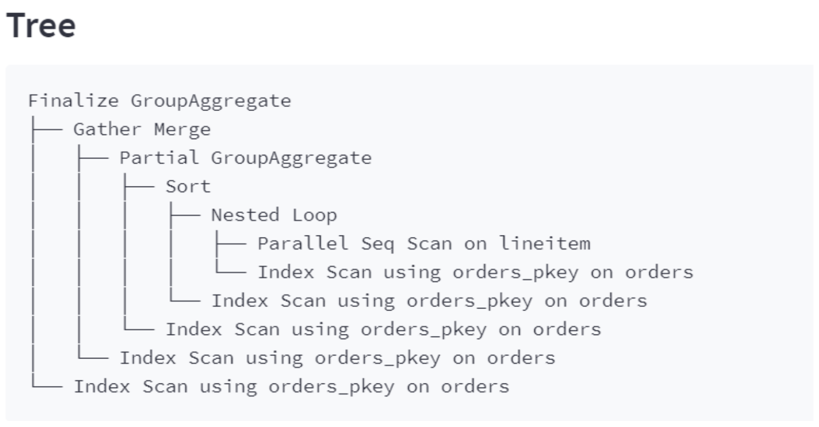


Query execution plan:

```
Step 1: Perform Parallel Seq Scan on lineitem  Filter: ((l_shipmode = ANY ('{MAIL,SHIP}'::bpchar[])) AND (l_commitdate < l_receiptdate) AND (l_shipdate < l_commitdate) AND (l
Step 2: Perform Index Scan using orders_pkey on orders  Index Cond: (o_orderkey = lineitem.l_orderkey)
Step 3: Perform Nested Loop
Step 4: Perform Index Scan using orders_pkey on orders  Index Cond: (o_orderkey = lineitem.l_orderkey)
Step 5: Perform Sort  Sort Key: lineitem.l_shipmode
Step 6: Perform Index Scan using orders_pkey on orders  Index Cond: (o_orderkey = lineitem.l_orderkey)
Step 7: Perform Partial GroupAggregate  Group Key: lineitem.l_shipmode
Step 8: Perform Index Scan using orders_pkey on orders  Index Cond: (o_orderkey = lineitem.l_orderkey)
Step 9: Perform Gather Merge  Workers Planned: 2
Step 10: Perform Index Scan using orders_pkey on orders  Index Cond: (o_orderkey = lineitem.l_orderkey)
Step 11: Perform Finalize GroupAggregate
```

Figure 4: Query Execution Steps

Example of query execution plan from example 3 detailing the steps which SQL performed for the input queries. You can scroll the text box.



Tree

```
Finalize GroupAggregate
├── Gather Merge
│   ├── Partial GroupAggregate
│   │   ├── Sort
│   │   │   ├── Nested Loop
│   │   │   │   ├── Parallel Seq Scan on lineitem
│   │   │   │   └── Index Scan using orders_pkey on orders
│   │   │   └── Index Scan using orders_pkey on orders
│   │   └── Index Scan using orders_pkey on orders
│   └── Index Scan using orders_pkey on orders
└── Index Scan using orders_pkey on orders
```

Figure 5: Visualisation of Query Execution Plan using Trees

A Query tree is also generated for visualisation purposes. The root node of the tree belongs to the top of the output. In this example, it will be ‘Finalize GroupAggregate’. The tree then branches out from the root node to ‘Gather Merge’ and ‘Index’ Scan’. The root node will be the last step to be executed in the query execution plan while the leaf nodes will be the first to be executed in this bottom up approach of the query execution plan.

Project Files

project.py

Main function of project.py is to run the interface. It is the __main__ module for the entire script. To run the interface, users have to execute streamlit run project.py in the terminal.

interface.py

interface.py module consists of functions for the GUI. These are some of the important functions for page interaction:

```
def main_interface()
```

The main interface function is the parent function which encapsulates all other GUI widgets. This function uses conditional operators to display data when an action is taken.

```
def sidebar():
```

The sidebar as seen is used to collect authentication variables for postgres authentication. After the variables are collected, the data is passed to preprocessing.py for authentication.

```
def example_query():
```

example_query function displays the ‘example” buttons as shown in figure two, when the button is pressed, the example query will be displayed in the text box and at the same time be executed as a query in the database server.

preprocessing.py

preprocessing.py is used as the back end of the system where the connection to database and calling of query is being executed here.

```
class Authentication:
    def __init__(self, db, user, passcode, host = 'localhost', port =5432):
        self.database = db
        self.username = user
        self.password = passcode
        self.host = host
        self.port = port
```

The Authentication class is used to store database authentication variables passed in through the side bar function in the interface module. By default, host and port has a default parameter unless otherwise specified in the sidebar.

```
def connect_sql(login):
```

Connect_sql function is used to check connection status to sql. If the connection is successful the function will return True, which allows the rest of the main interface to be loaded.

```
def read_query(query,login):
    query_explain = "explain " + query.replace("\n", " ")
```

read_query function is used to retrieve the query execution plan from the sql database. Whenever an sql query is being recorded in the text box, the read sql query will add “explain” to the front of the query string and then passed into the sql database. Once postgres sql returns the qep the data will then be passed into annotation.py module to restructure the query plan.

annotation.py

This file consists of codes to generate the annotations. After getting the query execution plan from the cursor, the query execution plan is then converted to QepNode objects and stored in a tree using the anytree library.

```
class QepNode(NodeMixin): # Add node feature
    def __init__(self, list_of_tuple, parent=None, children=None):
        self.list_tup = list_of_tuple
        self.explanation = self.get_info()
        self.name = self.get_node_type()
        self.parent = parent

        if children:
            self.children = children
```

The class QepNode initializes the node type of the query, the specifics of the node type and references to parent and child objects.

After storing the nodes in a tree, annotation will be generated from these QepNode objects. To generate the annotations, the tree is traversed using the Postorder traversal method - processing all nodes of a tree by recursively processing all subtrees, then finally processing the root node. This allows the steps to be read in the order of the bottom up query execution plan approach.

<u>Contribution by team members</u>

Our group has decided to split the project into 2 parts:

1. GUI
2. Query execution plan transformation
3. Report

Below is a table of the contributions of each group member.

Project Description	Assigned to
GUI	William, Qixiang
Query execution plan transformation	Jeremy, Tziyong
Report	Tziyong, Qixiang, Jeremy, William