

**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**

---

**SINGAPORE**

## **CZ4031 Database System Principles**

### **Project 2 Report**

### **Group 16**

**Mukizhitha Rajkumar (U2020288H)**

**Sherrie Quek Jie Ru (U2021010D)**

**To Chloe Pui (N2202501E)**

**Wang Kangxin (U1920039F)**

**Yong Hou Zhong (U2022811G)**

# Table of Contents

<b>Table of Contents</b>	<b>2</b>
<b>Contribution</b>	<b>3</b>
<b>Libraries used</b>	<b>4</b>
<b>Algorithm</b>	<b>5</b>
Main flow of the algorithm	5
PostgreSQL EXPLAIN	5
authenticate_connect(username, password, database_name)	6
connect()	6
runQuery(text)	6
getAQP(text)	6
annotate()	7
<b>GUI</b>	<b>8</b>
Database Login	8
Query Input	9
Annotated Query Execution Plan	10
View QEP Tree	10
Clear Text	10
Alternative Query Plans	11
<b>Software Limitations</b>	<b>12</b>

# Contribution

Team Member	Contribution
Mukizhitha Rajkumar	preprocessing.py, interface.py, report
Sherrie Quek Jie Ru	preprocessing.py, interface.py, report
To Chloe Pui	Interface.py, report
Wang Kangxin	annotation.py, interface.py, report
Yong Hou Zhong	annotation.py, preprocessing.py, report

# Libraries used

The libraries used for this GUI application are:

1. Psycopg2
  - Psycopg2 is a popular PostgreSQL database adapter for the Python programming language. Its main features are the complete implementation of the Python DB API 2.0 specification and thread safety (several threads can share the same connection).
2. Requests
  - Request library is the de facto standard for making HTTP requests in Python. We can use it to pass parameters and handle different request types
3. Selenium
  - Selenium is an open-source tool that automates web browsers
4. Tkinter
  - Tkinter is the de facto standard GUI library for Python and is included in all standard Python Distribution.

# Algorithm

## Main flow of the algorithm

- 1) Obtain valid SQL query from the user input
- 2) runQuery() in preprocessing.py will be called to generate QEP with the default settings
- 3) getAQP will be called in runQuery() to generate 3 AQP with some changes to the default settings
- 4) After generating all the JSON files, startAnnotation in annotation.py will be called to read and process the JSON to produce step by step descriptions of what is happening before the output is generated

## PostgreSQL EXPLAIN

The generation of the query execution plan (QEP) is done with the help of PostgreSQL's in-built command, EXPLAIN. This plan would be in JSON format so that it can be passed to be viewed as a QEP Tree for easy visualisation. The content of this plan includes the type of operator used and the total cost of the QEP. An example of a query execution plan is shown in Figure 1.

```
"Plan": {  
  "Node Type": "Hash Join",  
  "Parallel Aware": false,  
  "Async Capable": false,  
  "Join Type": "Inner",  
  "Startup Cost": 6572.44,  
  "Total Cost": 52941.6,  
  "Plan Rows": 351754,  
  "Plan Width": 31,  
  "Actual Startup Time": 285.716,  
  "Actual Total Time": 1136.762,  
  "Actual Rows": 353761,  
  "Actual Loops": 1,  
  "Output": [  
    "orders.o_orderkey",  
    "orders.o_orderdate",  
    "customer.c_custkey",  
    "customer.c_name"  
  ],  
  "Inner Unique": true,  
  "Hash Cond": "(orders.o_custkey = customer.c_custkey)",  
}
```

Figure 1: Example of QEP in JSON format

## `authenticate_connect(username, password, database_name)`

The main objective is to authenticate the connection to the PostgreSQL database and server. Three parameters that the user entered on the GUI are passed into the function, username, password for the server, and database name respectively. When a user presses the Login button on GUI, a connection with the server is established using the inbuilt `connect()` function from `psycopg2` library. After connecting to the server successfully, we save the login details in a JSON file (`logininfo.json`) that will be used when executing the query.

## `connect()`

Using the `logininfo.json` created in `authenticate_connect`, we connect to the database to query from it in `runQuery()`.

## `runQuery(text)`

This function here has a parameter, `text`, which is user input. It will execute the input string with the help of the 'Explain' command and generates the Query Execution Plan (QEP) in JSON format as shown in Figure 1 into a folder named "actual\_queryplans". It would then call the function `getAQP(text)`.

## `getAQP(text)`

This function here has a parameter, `text`, which is user input. It will execute the input string with the help of the 'Explain' command and generates the Alternative Query Plans (AQP). In this function, we have decided to generate only 3 Alternative Query Plans (AQP) in JSON format into a folder named "actual\_queryplans". These 3 AQPs have 3 different settings and they are shown in Figure 2. The three settings are uniquely chosen as these 6 operators are what we learnt in the lecture. We wanted to show the user how the different plans would look if these operators were turned off when executing the query. It would also explain why the QEP generated is of the lowest cost based on the comparison among other AQPs.

```
[
  [ "ENABLE_NESTLOOP", "OFF" ],
  [ "ENABLE_SEQSCAN", "OFF" ]
],

[
  [ "ENABLE_HASHJOIN", "OFF" ],
  [ "ENABLE_BITMAPSCAN", "OFF" ]
],

[
  [ "ENABLE_INDEXSCAN", "OFF" ],
  [ "ENABLE_MERGEJOIN", "OFF" ],
]
```

Figure 2: Settings list for AQPs

## annotate()

This function describes the annotated QEP explanations based on the different operators. We extracted the Node Types based on the JSON files, and we explained why certain node types were preferred over another. For instance, Sequential Scan is chosen sometimes because of the low selectivity of the predicate, and Index Scan is sometimes chosen because of the high selectivity of the predicate. In each case, the operators are chosen based on the ‘conditions’ or ‘filters’ in the query. At the end of the QEP, the total cost of the QEP is also shown.

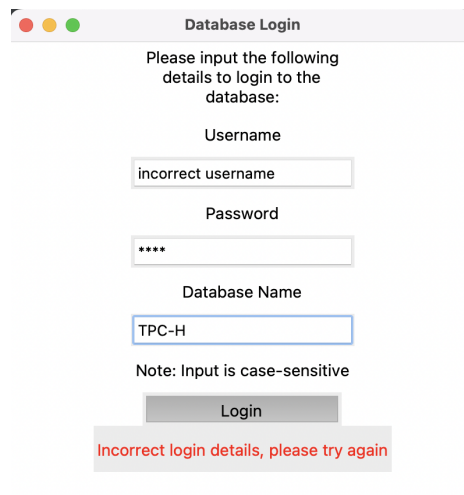
# GUI

## Database Login

Upon running the program, the 'Database Login' window will appear. To access the GUI, the user must connect to their local PostgreSQL server by correctly inputting the following (case-sensitive) information:

- Username
- Password
- Database name

Failure to input the correct details will not grant access to the GUI and lead to the display shown below in Figure 3.



The screenshot shows a window titled "Database Login" with a standard macOS-style title bar (red, yellow, green buttons). The window contains the following elements:

- A prompt: "Please input the following details to login to the database:"
- A label "Username" above a text input field containing the text "incorrect username".
- A label "Password" above a text input field containing four asterisks "\*\*\*\*".
- A label "Database Name" above a text input field containing the text "TPC-H".
- A note: "Note: Input is case-sensitive".
- A "Login" button.
- A red error message at the bottom: "Incorrect login details, please try again".

Figure 3: Login page



## Query Input

The user must input a query into the 'Input SQL Query Here' text area and click the 'Submit Query' button for the query to be executed. There are three possible scenarios regarding the query input that lead to different outputs in the 'Annotated Query Execution Plan' text area:

1. Nothing is submitted - the program has nothing to execute due to empty user input so the user is reminded to submit a query before clicking the 'Submit Query' button. This is shown in Figure 4.

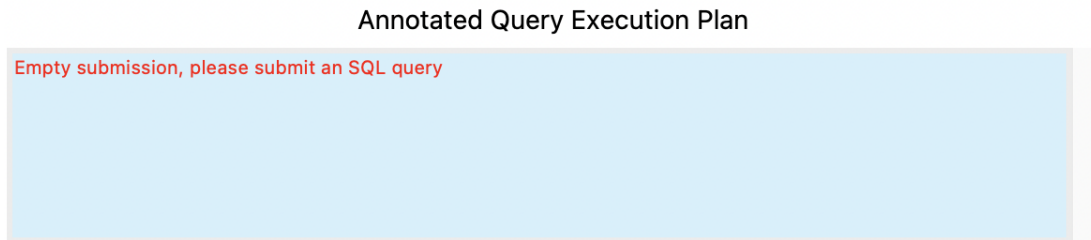


Figure 4: Empty user input

2. Invalid query is submitted - the program cannot recognise and execute the query and so the user is asked to check the query that they submitted, i.e. whether it ends with a ;. This is seen in Figure 5.

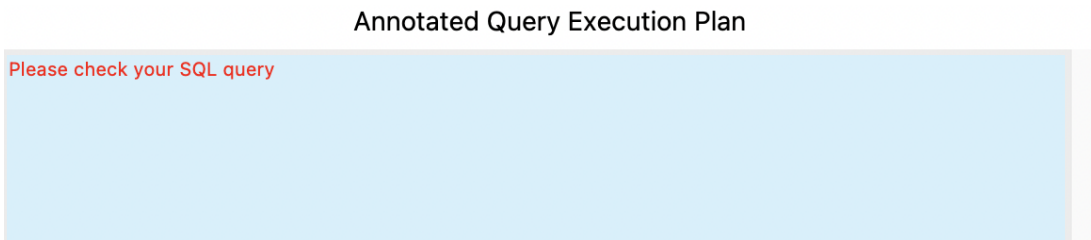


Figure 5: Invalid user input

3. Valid query is submitted - the program can execute the query and so the annotated query execution plan is displayed. This is seen in Figure 6.

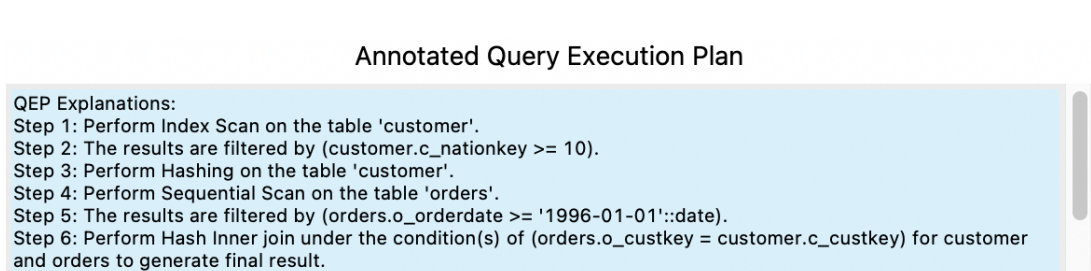


Figure 6: Valid user input

## Annotated Query Execution Plan

When the program is able to execute the submitted query, a step-by-step explanation of the execution plan is displayed (as shown in scenario 3 above). The execution and planning times along with the total cost of the query are shown.

Usage of the scrollbar allows for the full plan to be seen when necessary.

## View QEP Tree

If the user has submitted an executable query and clicks the ‘View QEP Tree’ button, the user will be led to “<https://tatiyants.com/>” which displays a visual representation of the QEP as seen in Figure 7.

This works by sending the query plan in JSON format to the website and the website will generate the QEP tree diagram.

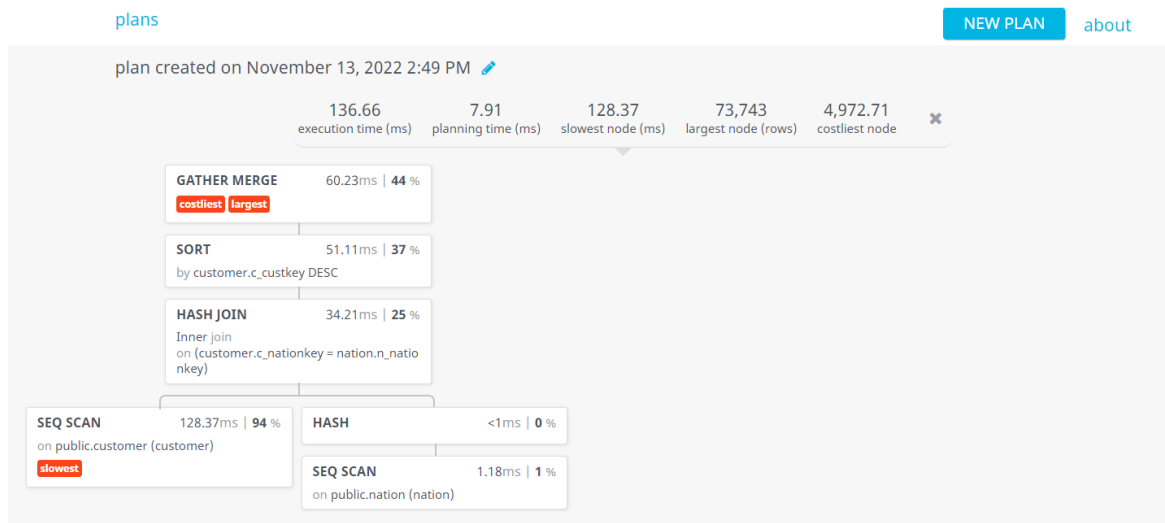


Figure 7: View QEP Tree

## Clear Text

There is a ‘Clear’ button that clears any text displayed in the panels, including the query the user input, once it is pressed. The user must use this “Clear” button before submitting the next query.

## Alternative Query Plans

Below the annotated QEP panel, there are three buttons, each of which leads to a visual representation of a different alternative query plan once pressed.

Figure 8 shows the first alternative AQP tree.

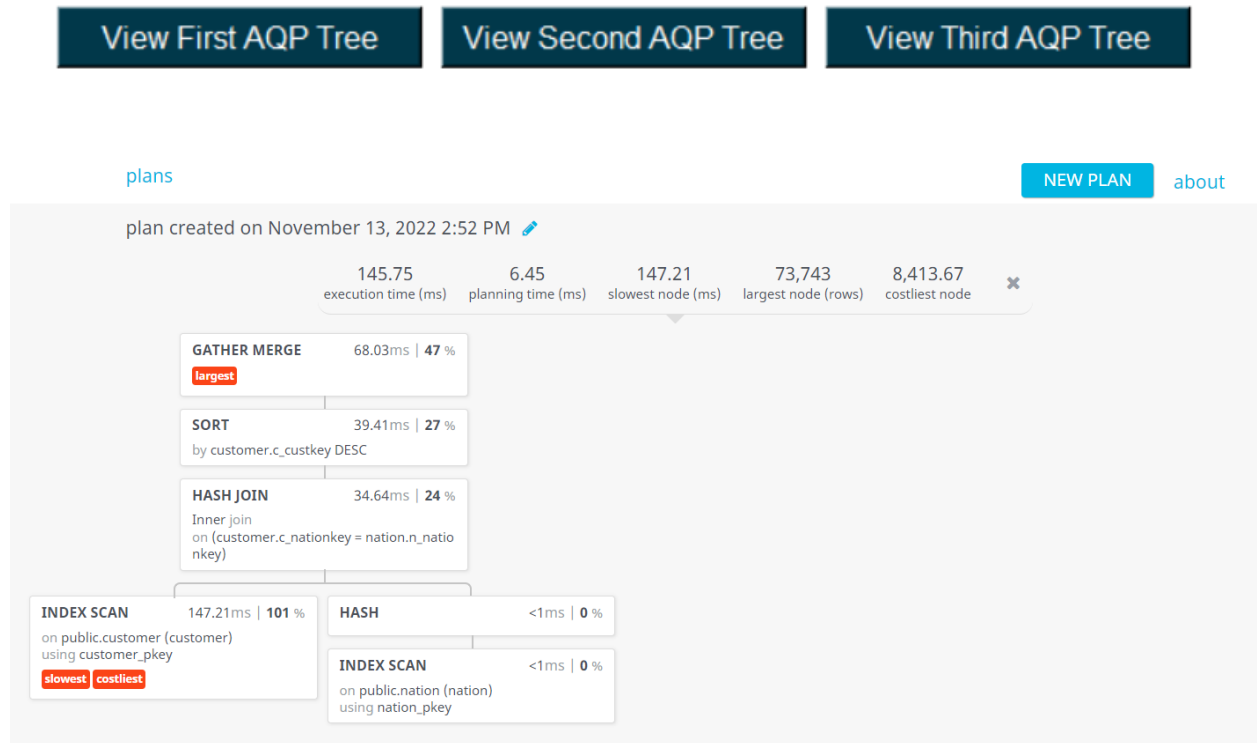


Figure 8: View First AQP Tree

## Software Limitations

- 1) Our software is unable to generate comparison explanations between the Query Execution Plan and the Alternative Query Plans.
- 2) Our software is limited to only explaining why certain scan and join operators are chosen.
- 3) Our software is only able to generate 3 AQPs with fixed settings instead of all the different combinations of settings. This is because as seen in our explanation for the settings under `getAQP(text)`, the settings were specially selected to highlight how certain operators being omitted would result in a larger cost. Though we also can let the user choose different settings on their own, we felt that it would be too overwhelming for the user.
- 4) User is unable to identify which settings we have changed for each AQP.
- 5) Our software requires the Microsoft Edge browser to view the QEP tree diagram. Please also wait for the visualisation to load.
- 6) User must press the “Clear” button after submitting 1 SQL query before they can submit a new SQL query.
- 7) Depending on the query type, it may take a while for our software to generate the results. Please only press each button once to avoid crashing the program.