SC3020 Database System Principles

AY23/24 Semester 1

Project 2 Report

Group 48

Team Members	
Atharv Gupta	U2123893E
Esther	U2122073H
Sim Jia Ming	N2304339G
Stephen	U2140974G

Acknowledgment

This project incorporates the use of a Large Language Model, specifically ChatGPT developed by OpenAI. We acknowledge and appreciate the contribution of OpenAI in providing access to this advanced language model, which has played a crucial role in assisting with the development of our user interface.

Individual Contributions

Name	Contributions
Atharv Gupta	1) Report
Esther	1) Implemented function to get user input for SQL query, table selection and blockID 2) Implemented visualisation of disk block content and access functions 3) Implemented UI frames for combined visualisation of part a) and b)
Sim Jia Ming	1) Implemented explore.py, project.py, database.py 2) Assisted with interface.py bugs fixes / implementation 3) Report - Installation guide, Architecture diagram, Key Functions and Algorithms
Stephen	1) Draw QEP Tree Functions 2) Display Node Details Functions 3) Integrated UI Functions

Objectives

There are two primary objectives of this project, initially it seeks a SQL query input from the user, upon receiving the input, the program creates a representation of the QEP in the form of an interactive operator tree, while also giving visual information about the disk blocks accessed while processing the query. Secondly it offers a user-friendly interface, which is the GUI component of the project, wherein the user can explore the contents of the disk blocks, by inputting the relation name and the block ID.

For this program, the host language has been chosen to be Python, while the Database Management System used is PostgreSQL. PostgreSQL is an open source relational database management system which in this project stores the TPC-H dataset, utilised for testing of the program.

The following sections of this report will give a basic idea of how this program has been implemented and the results displayed by it.

Implementation and Results

The implementation (as shown in figure 1) of this project has three major components which are:

- Interface: The GUI component, presenting a user friendly interface to input queries and observe the results
- Explore: The part of the code that explores the PostgreSQL database and fetches the information.
- Project: This is the executable file of the program.

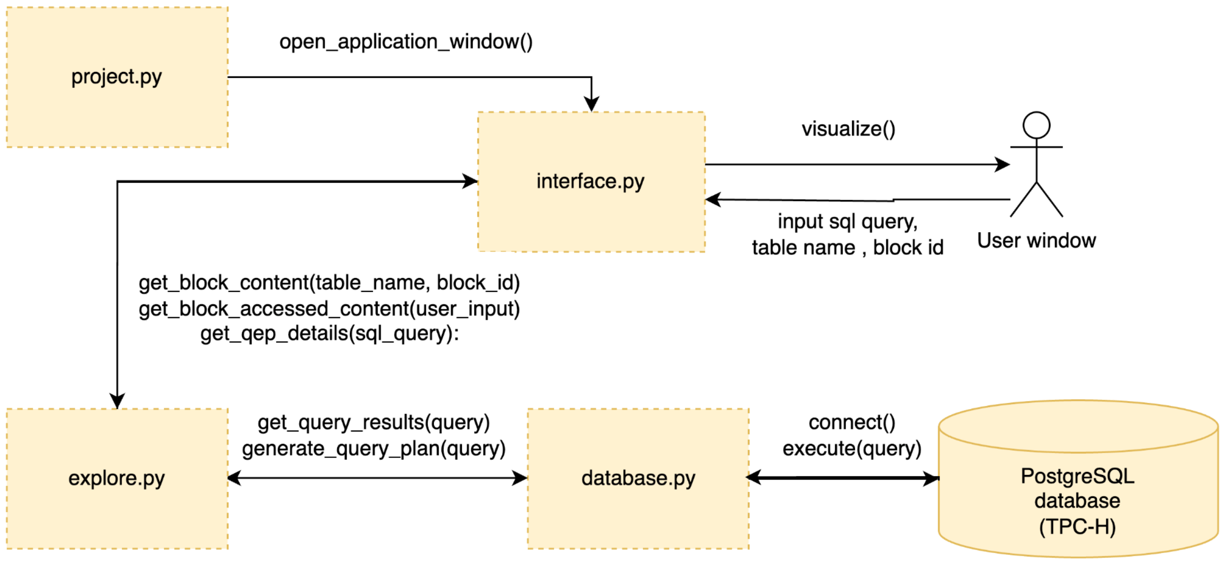


Figure 1. Architecture Diagram

User-Interface

Figure 2 illustrates the overview of the display a user would see after running the project.py file. The window would be titled “Database Block Visualisation” and will have five sections: Enter SQL Query, Block Accessed Information Table, Block Content, displaying the QEP and displaying the Node Details.

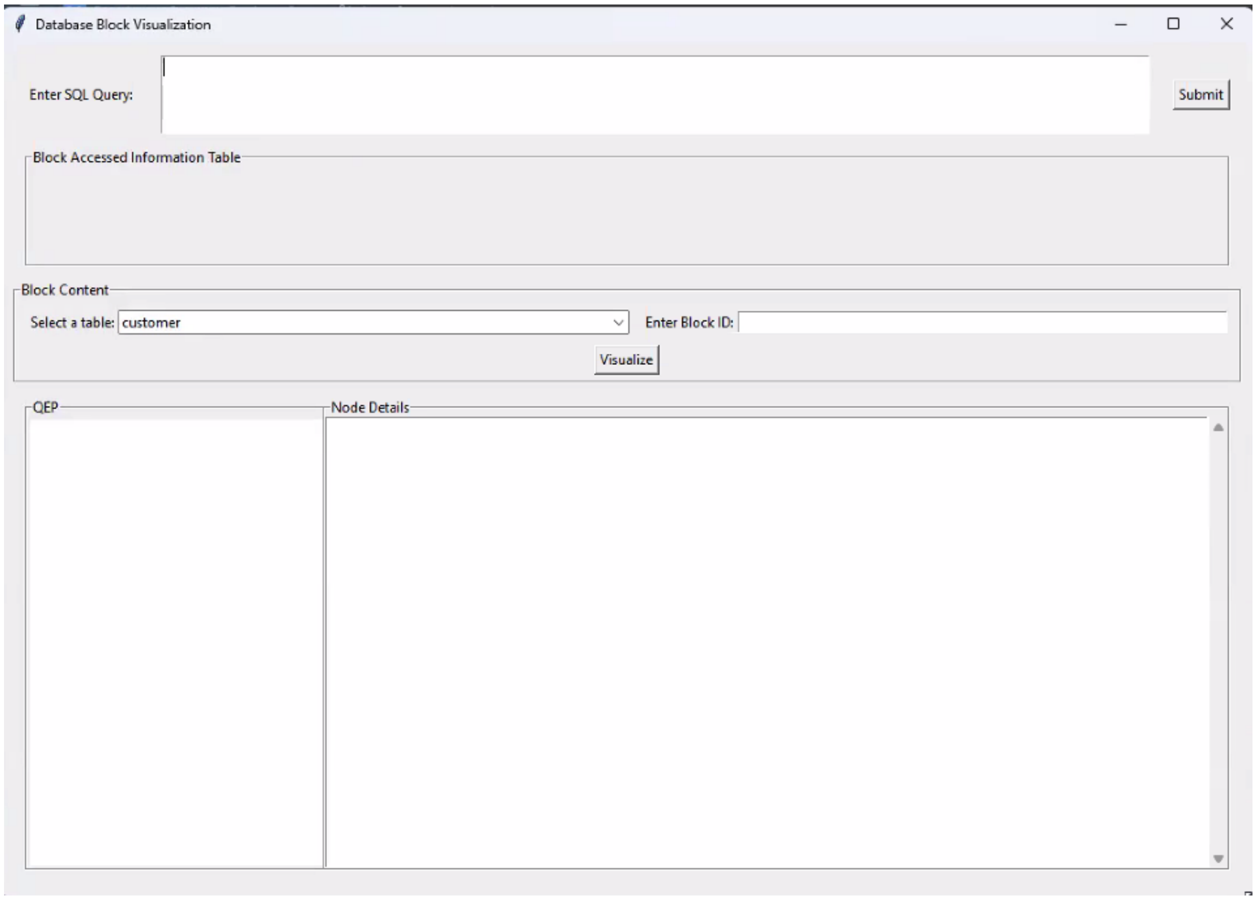
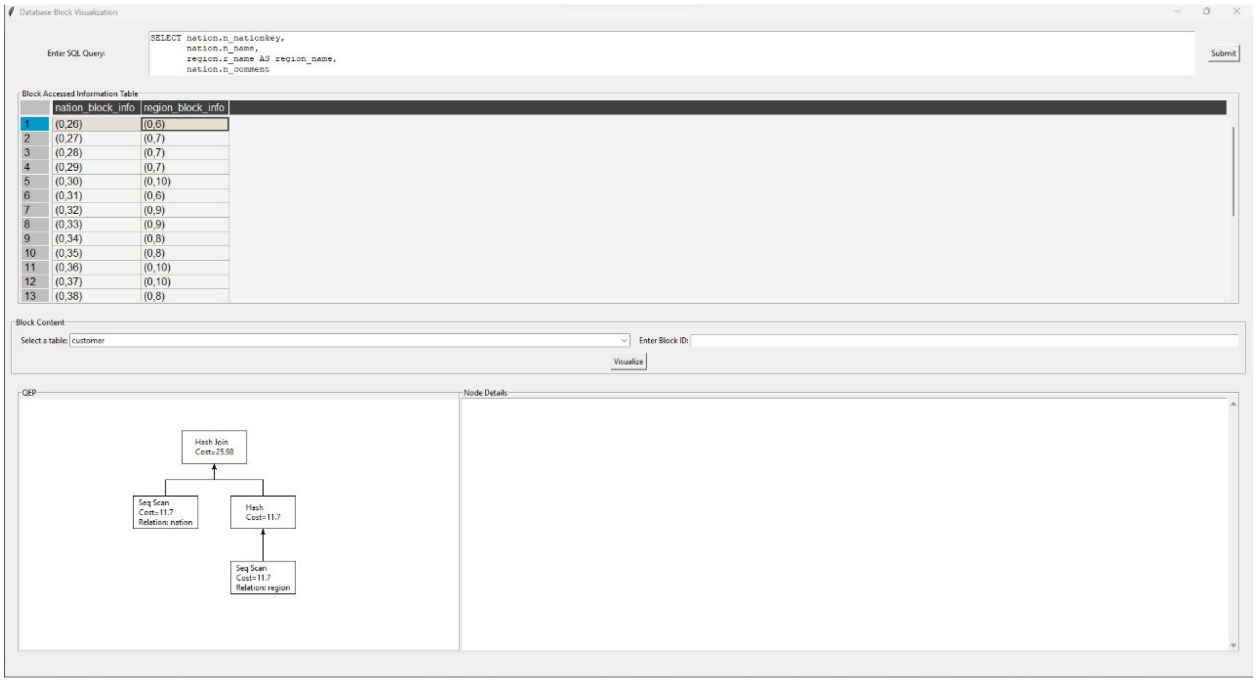


Figure 2: Overview of the Interface

- **SQL Query Input**

  The user is expected to populate the “Enter SQL Query” box by inputting an SQL query of their choice, as shown in figure 3. Upon entry and reviewing, the user can click on the Submit button to move forward.

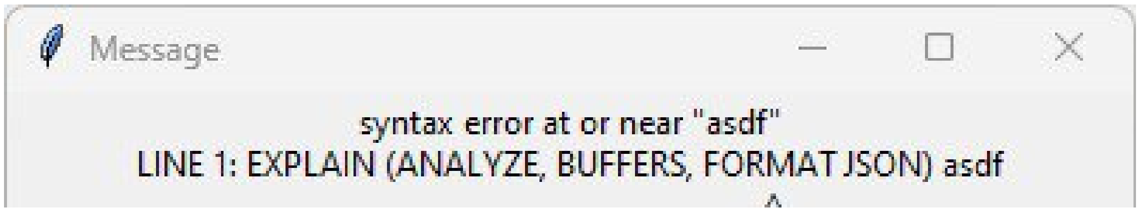


```
SELECT nation.n_nationkey,
       nation.n_name,
       region.r_name AS region_name,
       nation.n_comment
```

	nation_block_info	region_block_info
1	(0,26)	(0,6)
2	(0,27)	(0,7)
3	(0,28)	(0,7)
4	(0,29)	(0,7)
5	(0,30)	(0,10)
6	(0,31)	(0,6)
7	(0,32)	(0,9)
8	(0,33)	(0,9)
9	(0,34)	(0,8)
10	(0,35)	(0,8)
11	(0,36)	(0,10)
12	(0,37)	(0,10)
13	(0,38)	(0,8)

Figure 3: Entering SQL Query

  In case, the query inputted is not valid or accurate, an error prompt as seen in figure 4 will be displayed. The prompt will show the error that the user had committed and upon dismissing the prompt can re-enter the corrected SQL query.



```
syntax error at or near "asdf"
LINE 1: EXPLAIN (ANALYZE, BUFFERS, FORMAT JSON) asdf
```

Figure 4: Error Message Due to Invalid Query

- **Visualisation of the Operator Tree**

  Upon entry of a valid SQL query, the explore.py component of the program will process the query. Upon processing it will generate an interactive operator tree as seen in figure 5.

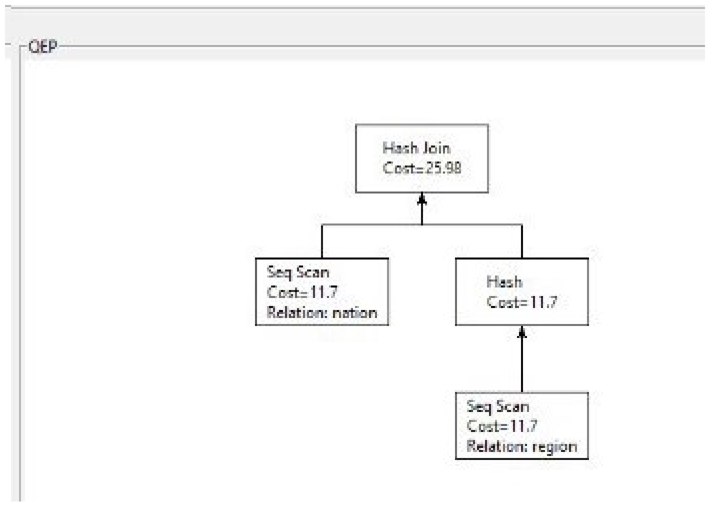
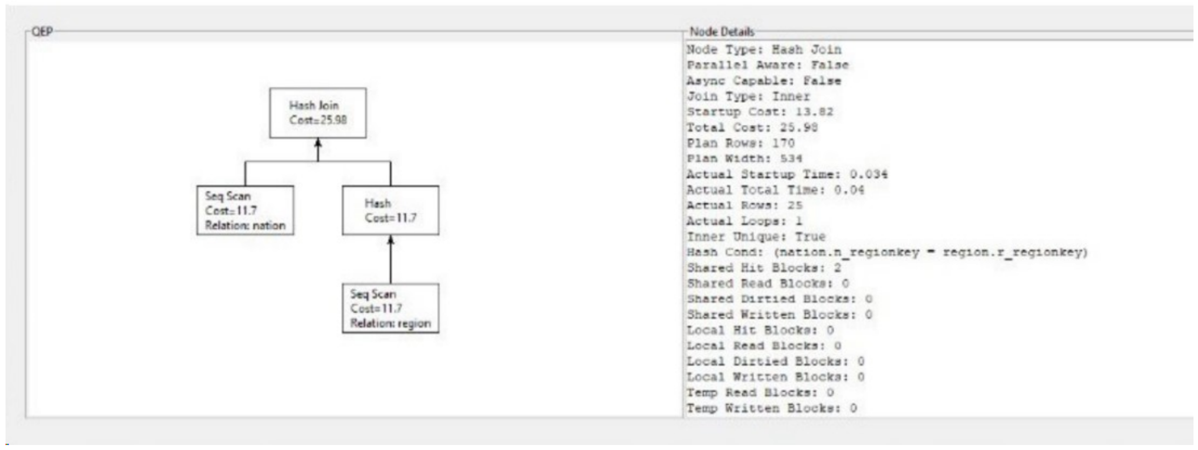


Figure 5: The QEP Operator Tree

The user can observe various operators, their ordering and execution during the query execution. The users can also click any node of the operator tree to get more information, as seen in figure 6.



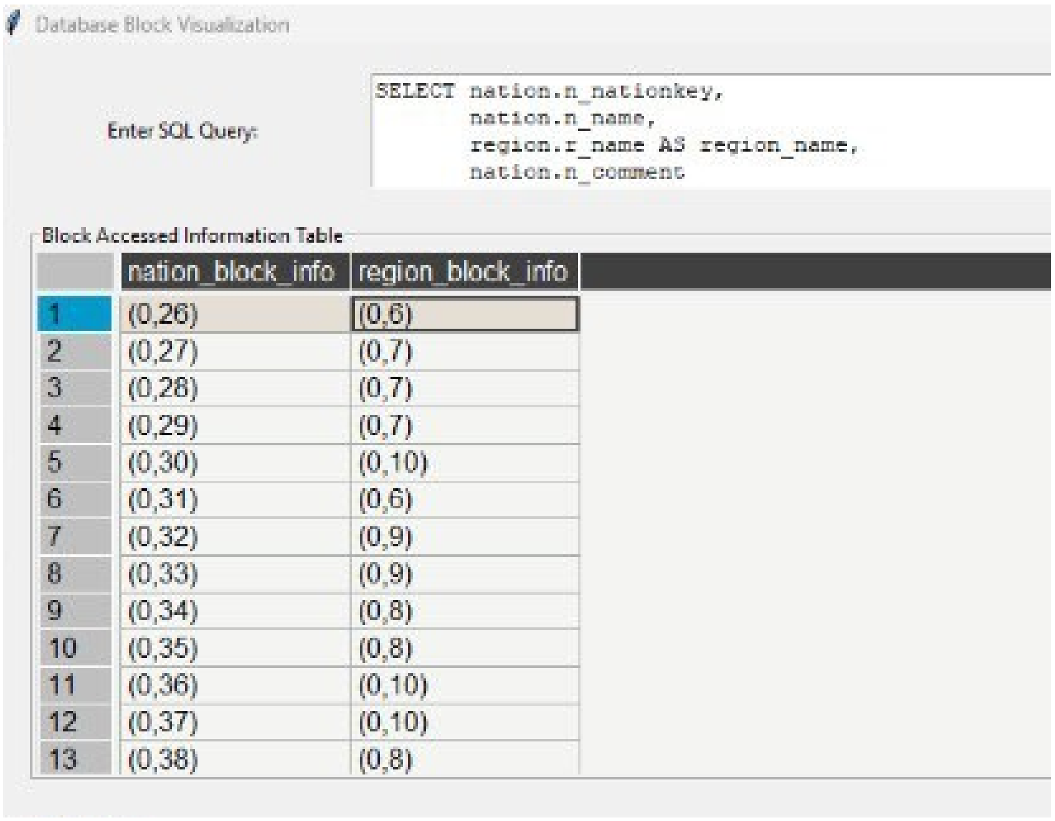
```
Node Type: Hash Join
Parallel Aware: False
Async Capable: False
Join Type: Inner
Startup Cost: 13.82
Total Cost: 25.98
Plan Rows: 170
Plan Width: 534
Actual Startup Time: 0.034
Actual Total Time: 0.04
Actual Rows: 25
Actual Loops: 1
Inner Unique: True
Hash Cond: (nation.n_regionkey = region.r_regionkey)
Shared Hit Blocks: 2
Shared Read Blocks: 0
Shared Dirtied Blocks: 0
Shared Written Blocks: 0
Local Hit Blocks: 0
Local Read Blocks: 0
Local Dirtied Blocks: 0
Local Written Blocks: 0
Temp Read Blocks: 0
Temp Written Blocks: 0
```

Figure 6: Node Details of Hash Join

In the “Node Detail” section, the user can gain further information about the operators used in every step, while also getting information about the number of disk blocks accessed, total cost and other relevant information such as buffer hits. This is the representation of QEP and lets the user see the internal working of how a database executes a query. Upon changing the initially inputted query the operator tree will be displaced by the new desired tree.

- **Block Accessed Information Table**

  The user also will be able to see the (block number, offset) pairs accessed by the query results in the "Block Accessed Information Table" section as seen in figure 7. These will be tabulated according to the table they belong to and are shown in this section.


Database Block Visualization
Enter SQL Query:
Block Accessed Information Table

```
SELECT nation.n_nationkey,
       nation.n_name,
       region.r_name AS region_name,
       nation.n_comment
```

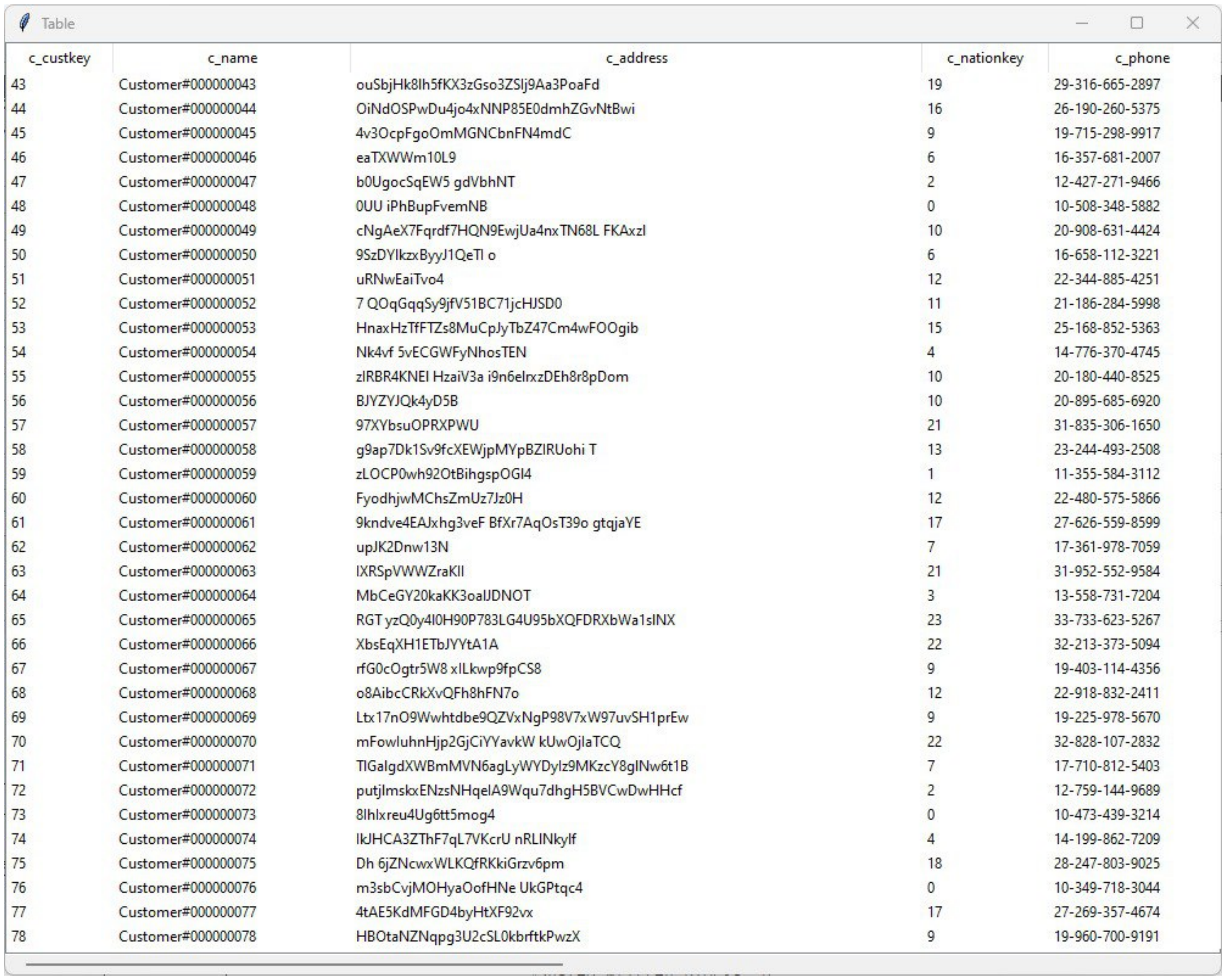
	nation_block_info	region_block_info
1	(0,26)	(0,6)
2	(0,27)	(0,7)
3	(0,28)	(0,7)
4	(0,29)	(0,7)
5	(0,30)	(0,10)
6	(0,31)	(0,6)
7	(0,32)	(0,9)
8	(0,33)	(0,9)
9	(0,34)	(0,8)
10	(0,35)	(0,8)
11	(0,36)	(0,10)
12	(0,37)	(0,10)
13	(0,38)	(0,8)

Figure 7: Block Accessed Information Table

So for example, the first tuple which shows (0,26) in the nation table and (0,6) in the region table are the block and tuple pair of the results of the query inputted by the user.

- **Visualisation of Disk Blocks**

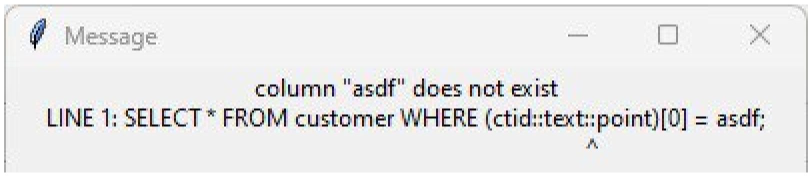
  The “Block Content” section of the interface, would allow the user to select a table name from a drop down list and enter the block ID to view the contents of that particular block. Upon entry and review, the user can click the “Visualise” button to view all the tuples situated in that block in the form of a pop up window as seen in figure 8.



c_custkey	c_name	c_address	c_nationkey	c_phone
43	Customer#000000043	ouSbjHk8lh5fKX3zGso3ZSIj9Aa3PoaFd	19	29-316-665-2897
44	Customer#000000044	OiNdOSPwDu4jo4xNNP85E0dmhZGvNtBwi	16	26-190-260-5375
45	Customer#000000045	4v3OcpFgoOmMGNCbnFN4mdC	9	19-715-298-9917
46	Customer#000000046	eaTXWWm10L9	6	16-357-681-2007
47	Customer#000000047	b0UgocSqEW5 gdVbhNT	2	12-427-271-9466
48	Customer#000000048	0UU iPhBupFvemNB	0	10-508-348-5882
49	Customer#000000049	cNgAeX7Fqrdf7HQN9EwjUa4nxTN68L FKAxzl	10	20-908-631-4424
50	Customer#000000050	9SzDYlkzxByyJ1QeTl o	6	16-658-112-3221
51	Customer#000000051	uRNwEaiTvo4	12	22-344-885-4251
52	Customer#000000052	7 QOqGqqSy9jfV51BC71jcHJSD0	11	21-186-284-5998
53	Customer#000000053	HnaxHzTfFTZs8MuCpJyTbZ47Cm4wFOOgib	15	25-168-852-5363
54	Customer#000000054	Nk4vf 5vECGWFyNhosTEN	4	14-776-370-4745
55	Customer#000000055	zIRBR4KNEI HzaiV3a i9n6elrxzDEh8r8pDom	10	20-180-440-8525
56	Customer#000000056	BJYZYJQk4yD5B	10	20-895-685-6920
57	Customer#000000057	97XYbsuOPRXPWU	21	31-835-306-1650
58	Customer#000000058	g9ap7Dk1Sv9fcXEWjpMYpBZIRUohi T	13	23-244-493-2508
59	Customer#000000059	zLOCP0wh92OtBihgspOGl4	1	11-355-584-3112
60	Customer#000000060	FyodhjwMChsZmUz7Jz0H	12	22-480-575-5866
61	Customer#000000061	9kndve4EAJxhg3veF BfXr7AqOsT39o gtqjaYE	17	27-626-559-8599
62	Customer#000000062	upJK2Dnw13N	7	17-361-978-7059
63	Customer#000000063	IXRSpVWWZraKII	21	31-952-552-9584
64	Customer#000000064	MbCeGY20kaKK3oalJDNOT	3	13-558-731-7204
65	Customer#000000065	RGT yzQ0y4l0H90P783LG4U95bXQFDRXbWa1sINX	23	33-733-623-5267
66	Customer#000000066	XbsEqXH1ETbJYYtA1A	22	32-213-373-5094
67	Customer#000000067	rfG0cOgtr5W8 xILkwp9fpCS8	9	19-403-114-4356
68	Customer#000000068	o8AibcCRkXvQFh8hFN7o	12	22-918-832-2411
69	Customer#000000069	Ltx17nO9Wwhtdbe9QZVxNgP98V7xW97uvSH1prEw	9	19-225-978-5670
70	Customer#000000070	mFowIuhnHjp2GjCiYYavkW kUwOjIaTCQ	22	32-828-107-2832
71	Customer#000000071	TlGalgdXWBmMVN6agLyWYDylz9MKzcY8gINw6t1B	7	17-710-812-5403
72	Customer#000000072	putjlmskxENzsNHqelA9Wqu7dhgH5BVCwDwHHcf	2	12-759-144-9689
73	Customer#000000073	8IhIxreu4Ug6tt5mog4	0	10-473-439-3214
74	Customer#000000074	IkJHCA3ZThF7qL7VKcrU nRLINkylf	4	14-199-862-7209
75	Customer#000000075	Dh 6jZNcwxWLKQfRKkiGrzv6pm	18	28-247-803-9025
76	Customer#000000076	m3sbCvjMOHyaOofHNe UkGPtqc4	0	10-349-718-3044
77	Customer#000000077	4tAE5KdMFGD4byHtXF92vx	17	27-269-357-4674
78	Customer#000000078	HBOtaNZNqpg3U2cSL0kbrftkPwzX	9	19-960-700-9191

Figure 8: Visualisation of Disk Block 1 of customer table

If a user enters an invalid Block ID an error would be shown as in figure 9. Upon changing the inputted values, the initial set of tuples would be replaced by the desired table.



```
column "asdf" does not exist
LINE 1: SELECT * FROM customer WHERE (ctid::text::point)[0] = asdf;
                                                              ^
```

Figure 9: Error Message for Invalid Block ID

Accessory Files

- The project.py is the executable file of the program. It executes the software, by invoking necessary functions, handling input/output of the queries and visualisation information.
- The database.py on the other hand, ensures connectivity of the program with the PostgreSQL DBMS.
- The test.py is a file made for testing this program.

Key Functions and Algorithms

draw_nodes_recursively

This function is used to visually represent a QEP in a tree structure on the canvas. It extracts key node information like type, cost, and relation name from the QEP in JSON format. Each node is drawn as a rectangle on the canvas and clicking on the node would trigger an event to display the detailed information of the node, thus allowing the user to explore and visualise different aspects of the QEP. If a node has a parent it would connect the child nodes to their parents with a line. Using a depth-first search approach, the function would be recursively called if the current node has child nodes, allowing for an interactive visualisation of the QEP.

get_qep_details

This function returns the QEP in JSON format. Given an SQL query user input for example `SELECT * FROM region`, we prepend `EXPLAIN (ANALYZE, BUFFERS, FORMAT JSON)` to the SQL user input so that we can get the query execution plan from PgSQL in JSON format.

get_block_content

Given the table name and block id from user input, this function returns a table containing all the tuples in that table and block requested by the user. Using the table name and block id, we craft the following query `SELECT * FROM {table name} WHERE (ctid::text::point)[0] = {block id}` to retrieve the block content.

get_block_accessed_content

This function returns the (block number, offset) pairs accessed by the query results as explained in the “Block Accessed Information Table” section. Given an SQL query input from the user, we first process the input and check if there is any SQL injection.

Next, for the ease of parsing later, we convert the query to lowercase, remove all trailing spaces, semicolons, excess white spaces, new lines and tabs.

Subsequently, we check if this process input contains nested queries as this function does not support inputs with nested queries (limitation).

Following up, we removed all the `GROUP BY, HAVING` and all the aggregation functions `(SUM, AVY, MIN, MAX, COUNT)` as they affect the results shown in the Block Accessed Information Table.

Afterwards, we scanned the query to look for all the relations involved and all their respective `ctid` columns into the query.

Lastly, we crafted the query which is then executed to get the table of blocks accessed by the results of the user SQL query input.

Example user input:

```
SELECT * FROM region;
```

Example crafted query:

```
SELECT region_block_info FROM (SELECT region.ctid as region_block_info, * FROM region) AS results_table;
```

detect_injection

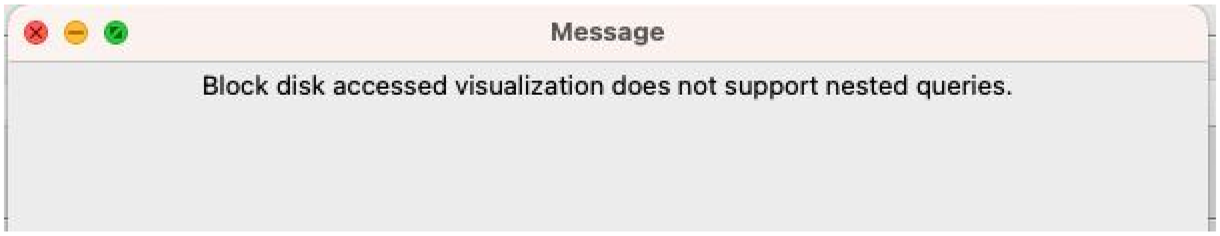
This function is designed to detect and prevent CUD (Create, Update, Delete) types of SQL injection attempts. SQL injection is a technique where an attacker inserts malicious SQL code into a query, potentially leading to unauthorised access or manipulation of the database. We have blocked the following operations: `DELETE, UPDATE, INSERT, CREATE, DROP, ALTER, INDEX`.

Limitations

Our program has the following limitations:

1. **Disk-block access:** Our program cannot visualise all the disk blocks that have been accessed by an input query as get_block_accessed_content function only retrieves the disk blocks that contain the result of the query. This is a subset of all disk blocks that were accessed in order to get the result. However, one can view the block information of the results retrieved by the user query input with the help of the Block Accessed Information Table. Moreover, the user can view the number of disk blocks accessed upon clicking the root node of the operator tree.

2. **Block Accessed Information Table for Nested Queries:** Our program cannot display the information of the blocks accessed when the user enters a nested query. It would show an error message as shown in Figure 10


Message

Block disk accessed visualization does not support nested queries.

Figure 10: Error Message for Nested Queries

Installation guide

Source Code

- GitHub Repository - https://github.com/SimJM/Database-System-Project-2

Software Requirements

Before you start, make sure you have the following software installed:

- PyCharm - https://www.jetbrains.com/pycharm/download/
- Python 3.11 - https://www.python.org/downloads/
- PostgreSQL - https://www.postgresql.org/download/
- Git - https://github.com/git-guides/install-git

Installation Steps

Follow these steps to clone, configure, and run the project on your system:

Step 0: Set up PostgreSQL and create TPC-H

Follow the steps in the appendix of the project document.

Step 1: Clone the repository

Open your command prompt and run the following command to clone the project repository. Alternatively, you can simply download the zip file.

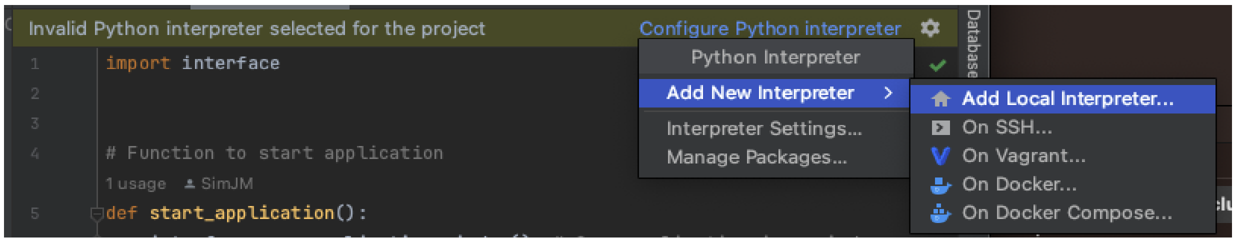
```
git clone https://github.com/SimJM/Database-System-Project-2.git
```

Step 2: Open the project in PyCharm

Launch PyCharm and open the `Database-System-Project-2` directory.

Step 3: Configure Python Interpreter

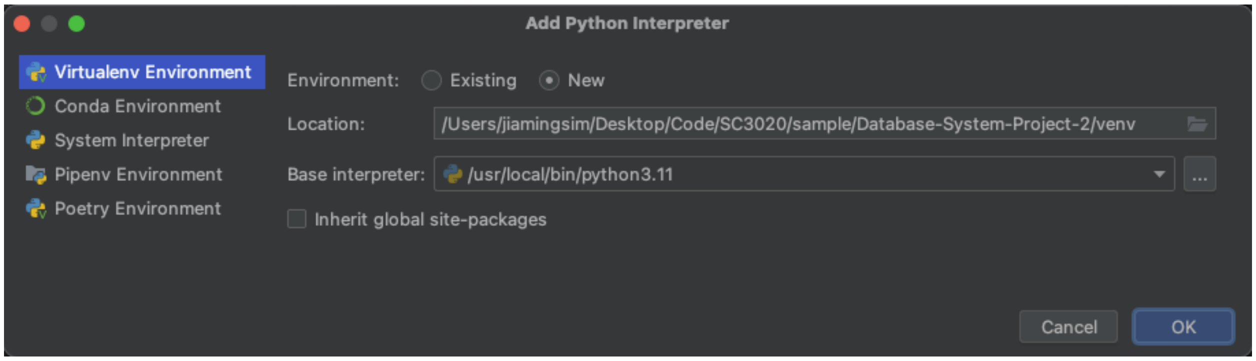
Open `project.py` then look for `Configure Python interpreter` to configure as shown below.



Invalid Python interpreter selected for the project

```
import interface


# Function to start application
1 usage  SimJM
def start_application():
```

Example configuration:



Environment: ( ) Existing [✓ New]

Location: /Users/jiamingsim/Desktop/Code/SC3020/sample/Database-System-Project-2/venv

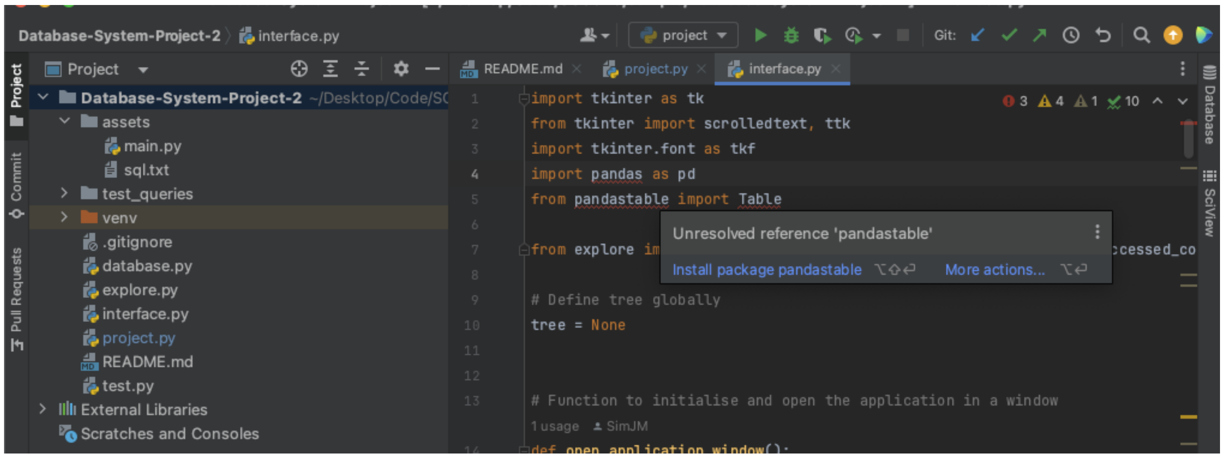
Base interpreter: /usr/local/bin/python3.11

[ ] Inherit global site-packages

Step 4: Install the relevant packages.

Open `project.py, explore.py, interface.py, database.py` and look for all the underlined imports in red. Right click on the underlined word then `install package <x>` as shown below.

Example:



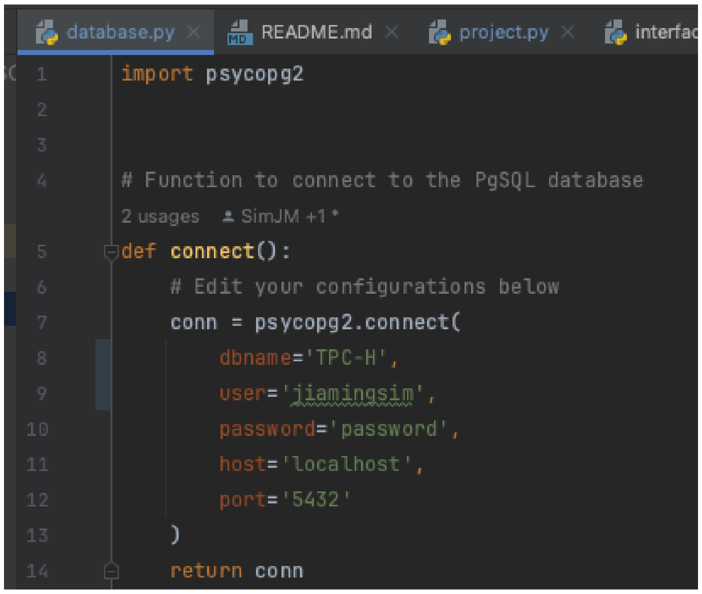
```
import tkinter as tk
from tkinter import scrolledtext, ttk
import tkinter.font as tkf
import pandas as pd
from pandastable import Table

from explore im                                    ccessed_co

# Define tree globally
tree = None


# Function to initialise and open the application in a window
1 usage  SimJM
def open_application_window():
```

Unresolved reference 'pandastable'

Install package pandastable ⌥⇧↩ More actions... ⌥↩

Step 5: Edit PgSQL database credentials

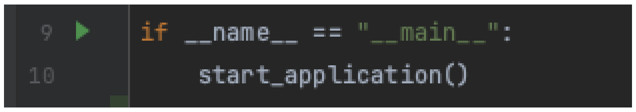
Open `database.py` and edit your PgSQL credentials.



```python
import psycopg2


# Function to connect to the PgSQL database
def connect():
    # Edit your configurations below
    conn = psycopg2.connect(
        dbname='TPC-H',
        user='jiamingsim',
        password='password',
        host='localhost',
        port='5432'
    )
    return conn
```

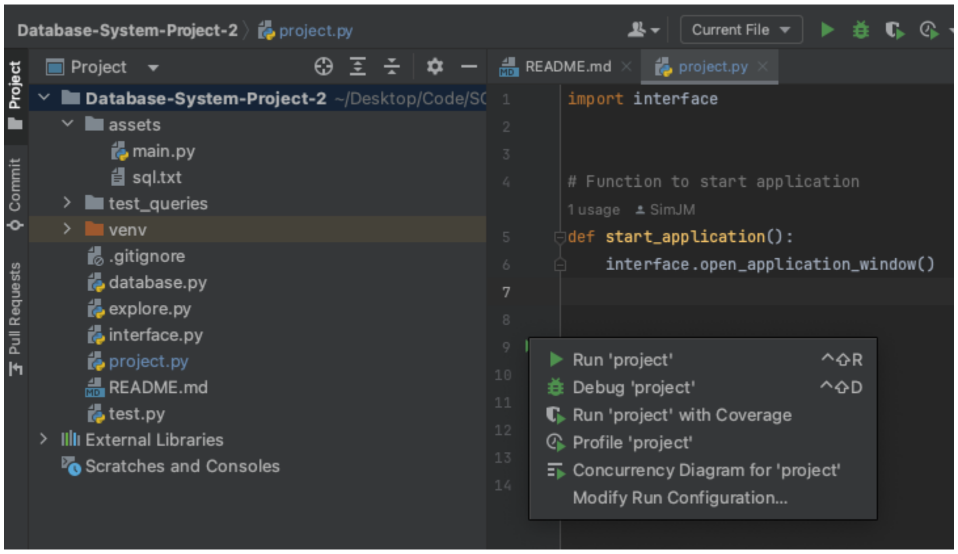
Step 6: Run the project

Once all the packages are installed, open `project.py` and look for the ‘play’ button in line 9.



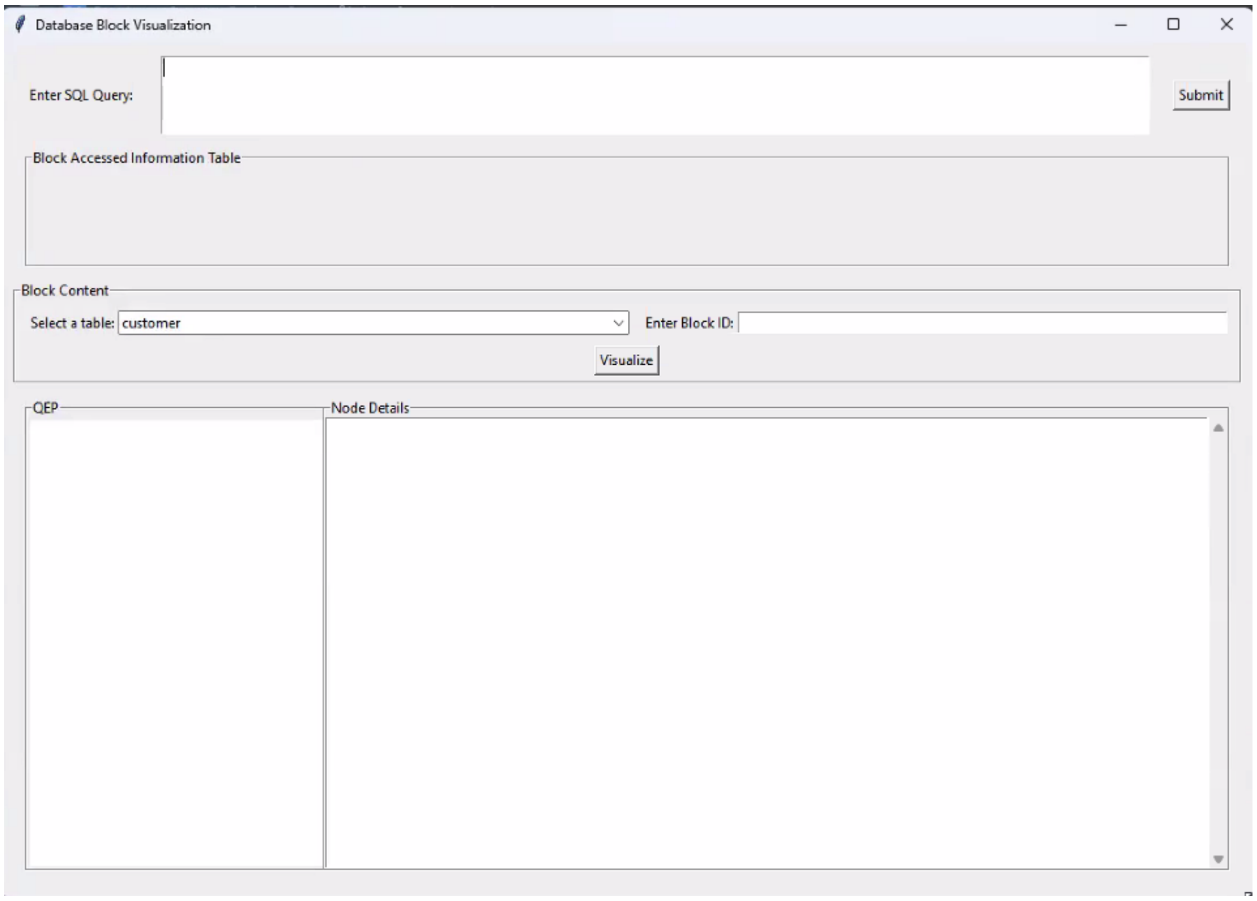
```python
 9  if __name__ == "__main__":
10      start_application()
```

Click `Run 'project'` as shown below.



```python
import interface


# Function to start application
def start_application():
    interface.open_application_window()
```

You should see the application open up in a window.


Database Block Visualization
Enter SQL Query:
Submit
Block Accessed Information Table
Block Content
Select a table: customer
Enter Block ID:
Visualize
QEP
Node Details