Database Management Tool

Implementation

This tool was developed in Python 3.12 using the *psycopg2* library for interfacing with SQL server. A prepared global statement provides all the database connection parameters and establishes the corresponding connection and cursor.

The main() function provides the menu interface to which all other functions may be accessed. Each functionality works similarly: Necessary user inputs are taken, processed (if necessary), and substituted into a format string to comprise a complete SQL query. This query is then fed to a helper function that delivers the query to SQL server and returns its status, including error messages, if any.

Documentation

Function Usage

For all functions, the input table is to be located in the *public* schema of SQL Server and entered without quotes. Likewise, every function will return a custom response on success (covered in their specific sections below) or the SQL Server's error response on failure.

In either case, you are presented the option to 1. Return to the function (for additional queries), 2. Return to the menu, or 3. Exit the program on query closure, as shown below.

```
1.Return
2.Return to menu
3.Exit
```

1. Insert Data

Enter the INSERT parameters as prompted. All values should adhere to their data types (strings in single '' quotes, for example), and they cannot be blank. Once all necessary inputs are gathered, the query will automatically be sent. If successful, the console will display "Record successfully inserted."

```
:1
Enter Table: processor
Enter values:
processor_brand: 'Intel'
processor_name: 'Core i3-2310E'
processor_gnrtn: '2nd'
Record successfully inserted.
```

2. Delete Data

Enter the DELETE parameters as prompted. The condition is to be entered as a SQL WHERE snippet of arbitrary composition and length, adhering to all of its syntax rules, and potentially allowing for nested queries if desired. It cannot be left blank. If successful, the console will display "Record(s) successfully deleted."

```
:2
Enter table: processor
Enter condition:processor_name = 'Missing'
Record(s) successfully deleted.
```

3. Update Data

Enter the UPDATE parameters as prompted. The new values should adhere to their data types, and they cannot be blank. The condition is again to be entered as a SQL WHERE snippet, which cannot be blank. If successful, the console will display "Record(s) successfully updated."

```
:3
Enter table: processor
Set new values:
processor_brand: 'Test'
processor_name: 'Test'
processor_gnrtn: 'Test'
Enter condition: processor_name = 'Missing'
Record(s) successfully updated.
```

4. Search/Sort Data

This functionality combines two into one. Enter the SELECT and/or ORDER BY parameters as prompted. Columns may be entered separated by commas, or a single asterisk (*) to signify the wildcard. Being as this is a SELECT query, the condition may be left blank. Likewise, both of the proceeding sorting conditions may be left blank as well if you only want a SELECT query. Otherwise, enter exactly one column and *ASC* or *DESC*, respectively. If successful, the console will display the matching tuples, one each line.

```
:4
Enter Table: processor
Columns: processor_brand, processor_name, processor_gnrtn
Select Column(s): *
Enter condition:
Enter sorting column: processor_name
Enter sorting order: ASC
[processor_brand, processor_name, processor_gnrtn]
('AMD', 'A6-9225 Processor', '10th')
('AMD', 'APU Dual', '10th')
('AMD', 'Athlon Dual', '10th')
('Intel', 'Celeron Dual', 'Missing')
('Intel', 'Core i3', '10th')
('Intel', 'Core i3', '11th')
('Intel', 'Core i3', '7th')
('Intel', 'Core i5', '10th')
```

5. Aggregate functions

Enter the parameters as prompted. Columns may be singular, a wildcard, or separated by commas. The aggregate function should be one of the valid SQL aggregates. If successful, the console will return the result.

```
:5
Enter table: model
Columns: brand, model, processor_name, processor_gnrtn, ram_gb, ram_type, ssd, hdd, os,
Enter aggregate function: AVG
Enter column: latest_price
[(Decimal('79697.336206896552'),)]
```

6. Joins

Enter the parameters as prompted. Although the tables are labeled ‘Left’ and ‘Right’, this is a normal JOIN operation, so the order does not matter. No tables may be left blank. The key must be a column shared by the two tables, and it cannot be blank. If successful, the console will display the matching tuples, one each line.

```
:6
[('model',), ('processor',)]
Enter Left Table: processor
Enter Right Table: model
Enter Key: processor_name
Select Column(s): *
[*]
('Intel', 'Celeron Dual', 'Missing', 'HP', '14a', 'Celeron Dual', 'Missing', '4 GB GB', 'DDR4', '0 GB', '512 GB', 'Windows', '64-bit',
('Intel', 'Core i3', '7th', 'HP', '14s', 'Core i3', '11th', '4 GB GB', 'DDR4', '256 GB', '0 GB', 'Windows', '64-bit', 0, 'ThinNlight',
('Intel', 'Core i3', '11th', 'HP', '14s', 'Core i3', '11th', '4 GB GB', 'DDR4', '256 GB', '0 GB', 'Windows', '64-bit', 0, 'ThinNlight',
('Intel', 'Core i3', '10th', 'HP', '14s', 'Core i3', '11th', '4 GB GB', 'DDR4', '256 GB', '0 GB', 'Windows', '64-bit', 0, 'ThinNlight',
```

7. Grouping

Enter the parameters as prompted. The grouped column must be singular, and it cannot be blank. If successful, the console will return the matching tuples and their count, one each line.

```
:7
Enter Table: processor
Columns: processor_brand, processor_name, processor_gnrtn
Select Column(s): processor_brand
Select Group Column: processor_brand
('M1', 1)
('AMD', 9)
('Intel', 19)
('Qualcomm', 1)
```

8. Subqueries

Enter the parameters as prompted. As their input fields indicate, the outer column represents the outer SELECT and the inner column represents the nested (inner) SELECT. Neither column may be blank. If successful, the console will display the matching tuples, one each line.

```
:8
Select outer column(s): latest_price, reviews
Enter inner table: model
Enter inner column(s): *
(26470, 23)
(42490, 561)
(42990, 152)
(58990, 134)
(46790, 325)
```

9. Transactions

Enter the corresponding key to perform an action. *Commit* will commit all successful queries performed this session (program runtime), and *Rollback* will perform a roll back. A successful commit will return "Transaction successfully committed.", and a successful roll back will return "Transaction successfully rolled back."

```
:9
1.Commit
2.Rollback
3.Return to menu
:1
Transaction successfully commited.
```

```
:9
1.Commit
2.Rollback
3.Return to menu
:2
Transaction successfully rolled back.
```

10. Error Handling

All error messages displayed by the console will be those sent by the SQL Server and should be handled as such. Common errors include:

Syntax errors,

```
:1
Enter Table: proce
Error: relation "public.proce" does not exist
LINE 1: SELECT * FROM public.proce LIMIT 0
                      ^
```

And abort errors when attempting to send another query following a failure. If met with this error, roll back the current transaction before sending any additional queries.

```
:1
Enter Table: processor
Error: current transaction is aborted, commands ignored until end of transaction block
```