STM_ICT Project 2021 SP2 #28 Simulate Sensor Data and Dashboard

Database Code Documentation

Team member:
YuxuanLi, YifeiRan, Magura, Munyaradzi

CONTEXT

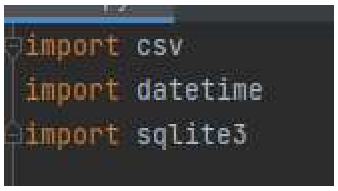
Overview

This document is used to describe the configuration and operation instructions of the database module in the project.

Resource

This module uses the SQLite module that comes with python. Therefore, users do not need to install the library before using this module, just confirm whether the library has been imported in the py file.
Choose SQLite as the project database type for the following reasons:
1. The main modules of the project are developed based on python. SQLite3 can be integrated with Python using the sqlite3 module.
2. SQLite is an in-process library. Unlike other databases, no user configuration is required in the system. At the same time, SQLite is also lightweight and has low system environmental requirements.
3. SQLite is self-sufficient, and the use of this database does not require users to generate external dependencies.

In addition, this module also uses the csv library and datetime library that comes with python.



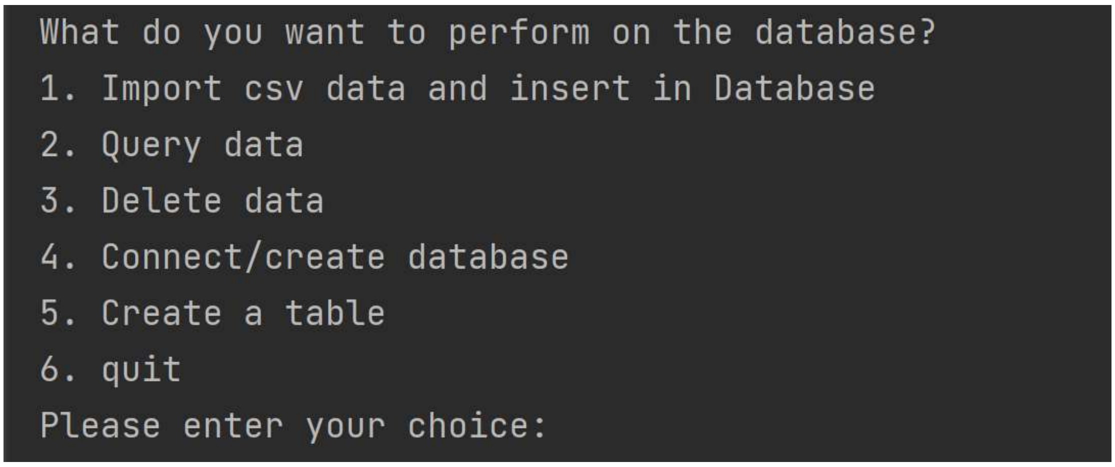
```
import csv
import datetime
import sqlite3
```

SQLite reference: https://www.sqlite.org/index.html
https://docs.python.org/3/library/sqlite3.html

User Interaction

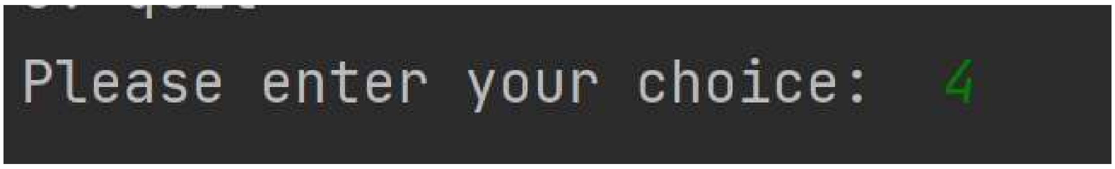
This module provides users with preset some project database operation methods. Users can import csv files exported by other modules into the database through simple keyboard input.

After running, py file users can see the following output in the running window:



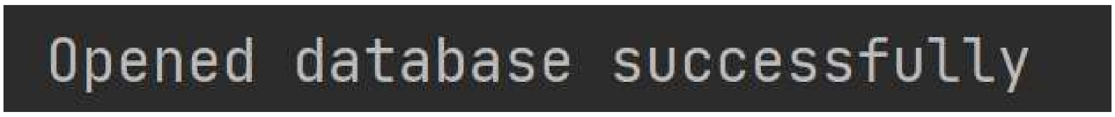
```
What do you want to perform on the database?
1. Import csv data and insert in Database
2. Query data
3. Delete data
4. Connect/create database
5. Create a table
6. quit
Please enter your choice:
```

If the user is using the module for the first time. Please enter "4" to create the database.



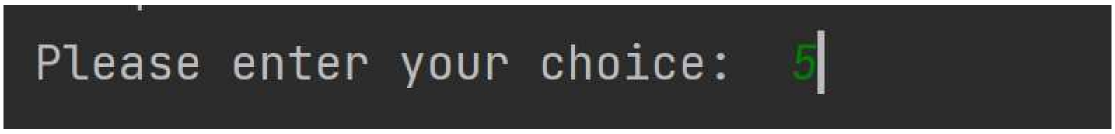
```
Please enter your choice:  4
```

If there is no database file named "sim_database.db" in the project path. The method will create a file in the project path.
Note: If there is a file with the same name, this method will only connect to the database with that name. It will not overwrite the file with the same name.
The following is a prompt for successful creation:

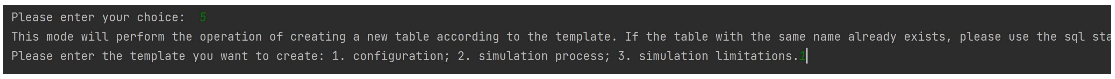


```
Opened database successfully
```

Create a table



```
Please enter your choice:  5
```

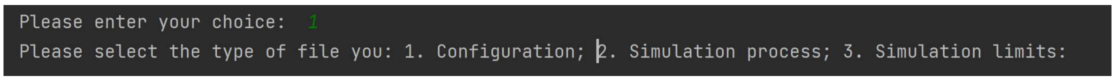


```
Please enter your choice:  5
This mode will perform the operation of creating a new table according to the template. If the table with the same name already exists, please use the sql sta
Please enter the template you want to create: 1. configuration; 2. simulation process; 3. simulation limitations.1
```

Before using the database, users need to create tables according to the template. After entering 5, the user enters "1", "2", and "3" respectively and press Enter to create a table based on the template in database.

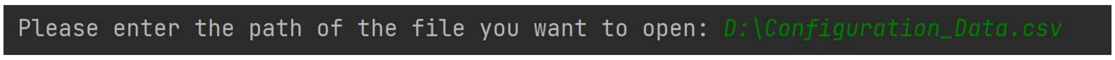
Import csv data

In the data import mode, users need to select the corresponding type according to the data they want to import. The wrong type selection will cause the import to fail.



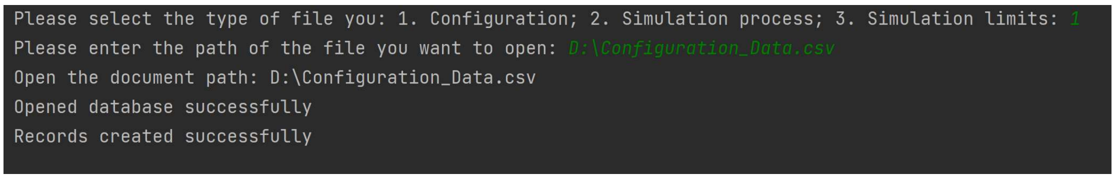
```
Please enter your choice:  1
Please select the type of file you: 1. Configuration; 2. Simulation process; 3. Simulation limits:
```

The user needs to enter the path to import the csv file.



```
Please enter the path of the file you want to open: D:\Configuration_Data.csv
```

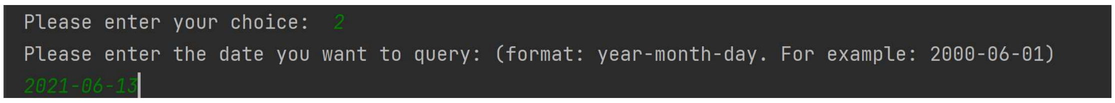
After the import is successful, the output box will display a message.



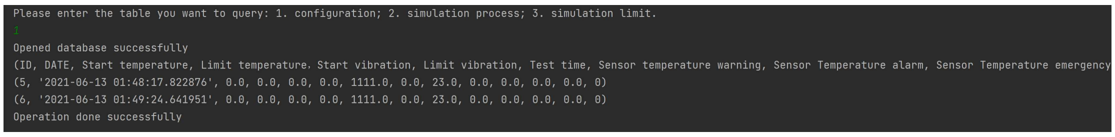
```
Please select the type of file you: 1. Configuration; 2. Simulation process; 3. Simulation limits: 1
Please enter the path of the file you want to open: D:\Configuration_Data.csv
Open the document path: D:\Configuration_Data.csv
Opened database successfully
Records created successfully
```

Query data

Users can query the eligible data in this mode based on the year, month, and day information. The format of year, month and day must meet the rule "year-month-day"



```
Please enter your choice:  2
Please enter the date you want to query: (format: year-month-day. For example: 2000-06-01)
2021-06-13
```

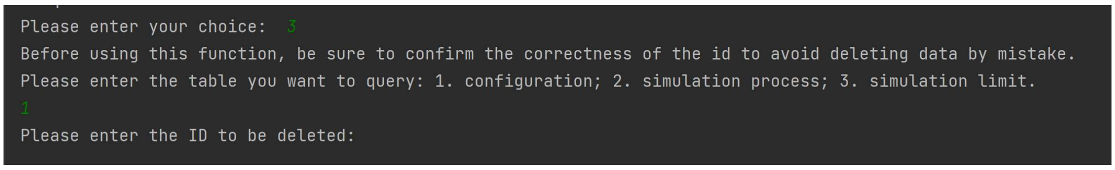


```
Please enter the table you want to query: 1. configuration; 2. simulation process; 3. simulation limit.
1
Opened database successfully
(ID, DATE, Start temperature, Limit temperature, Start vibration, Limit vibration, Test time, Sensor temperature warning, Sensor Temperature alarm, Sensor Temperature emergency
(5, '2021-06-13 01:48:17.822876', 0.0, 0.0, 0.0, 0.0, 1111.0, 0.0, 23.0, 0.0, 0.0, 0.0, 0.0, 0)
(6, '2021-06-13 01:49:24.641951', 0.0, 0.0, 0.0, 0.0, 1111.0, 0.0, 23.0, 0.0, 0.0, 0.0, 0.0, 0)
Operation done successfully
```

If only the title field is output, it means that the database does not have a record of that time.

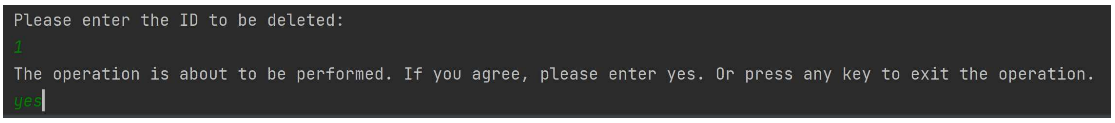
Delete data

Please use this function with caution, once it is executed, it cannot be restored.
Before the operation, you need to obtain the id of the data you want to delete (you can use the search function to obtain the method)

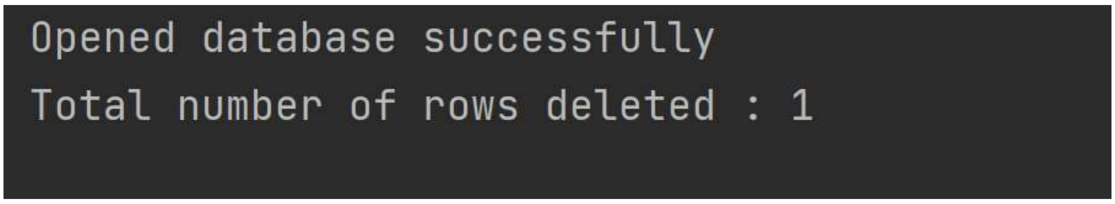


```
Please enter your choice:  3
Before using this function, be sure to confirm the correctness of the id to avoid deleting data by mistake.
Please enter the table you want to query: 1. configuration; 2. simulation process; 3. simulation limit.
1
Please enter the ID to be deleted:
```

Enter yes and press Enter to perform the operation.



```
Please enter the ID to be deleted:
1
The operation is about to be performed. If you agree, please enter yes. Or press any key to exit the operation.
yes
```

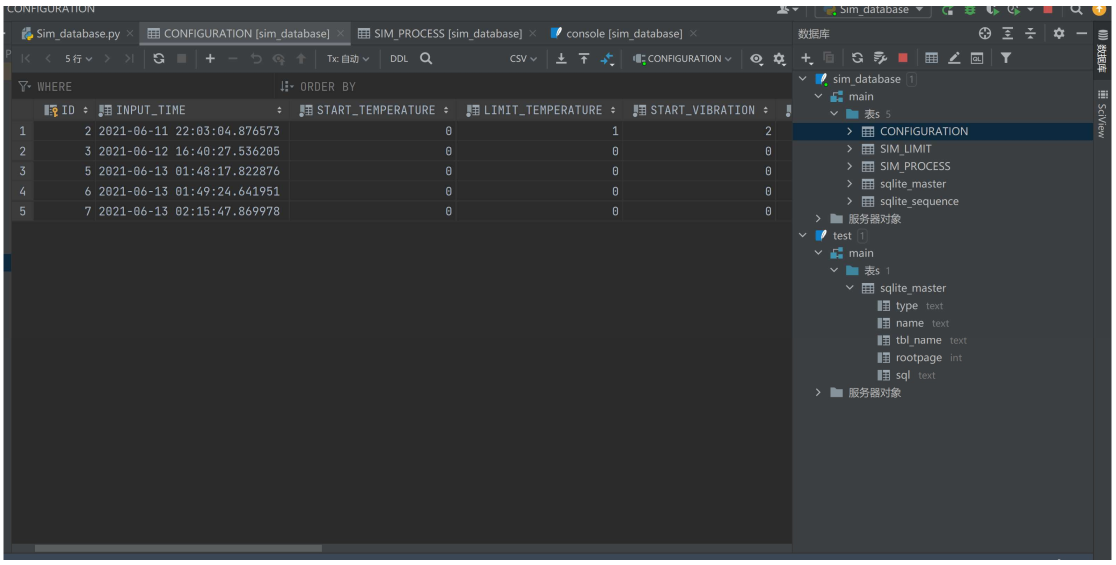


```
Opened database successfully
Total number of rows deleted : 1
```

Other methods of operating the database:

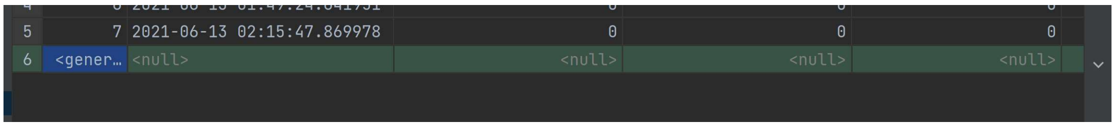
Pychram:

Using pychram, you can directly use the graphical interface to perform operations on each table in the database (such as adding, deleting, and modifying data).



	ID	INPUT_TIME	START_TEMPERATURE	LIMIT_TEMPERATURE	START_VIBRATION
1	2	2021-06-11 22:03:04.876573	0	1	2
2	3	2021-06-12 16:40:27.536205	0	0	0
3	5	2021-06-13 01:48:17.822876	0	0	0
4	6	2021-06-13 01:49:24.641951	0	0	0
5	7	2021-06-13 02:15:47.869978	0	0	0

Enter data directly on the graphical interface to add it to the table.



5	7	2021-06-13 02:15:47.869978	0	0	0
6	<gener…	<null>	<null>	<null>	<null>

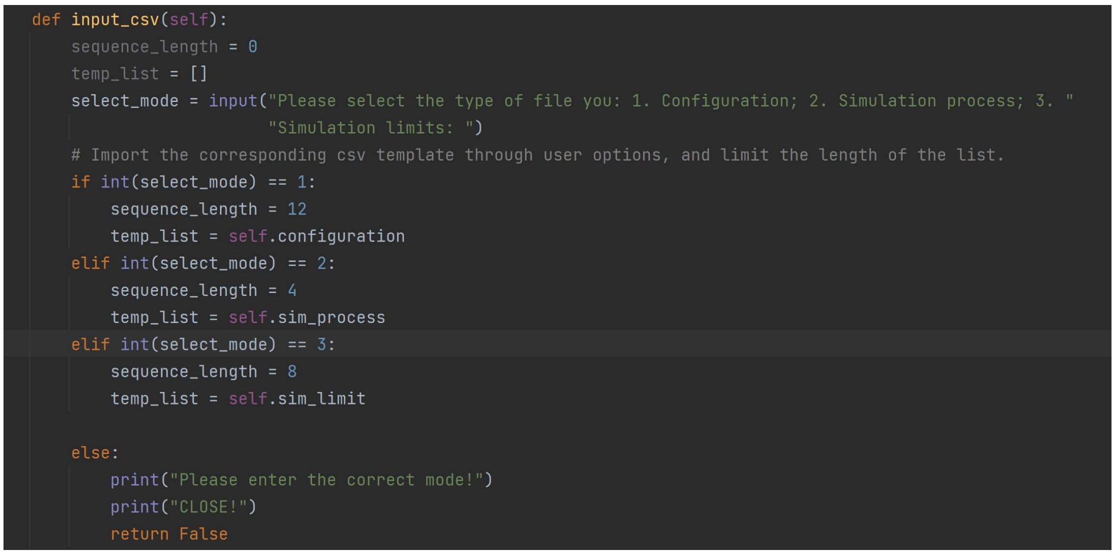
Console& Database operating software

In the pychram database console, you can directly use sql statements to operate the database. Users can also use software such as oracle to operate the database.

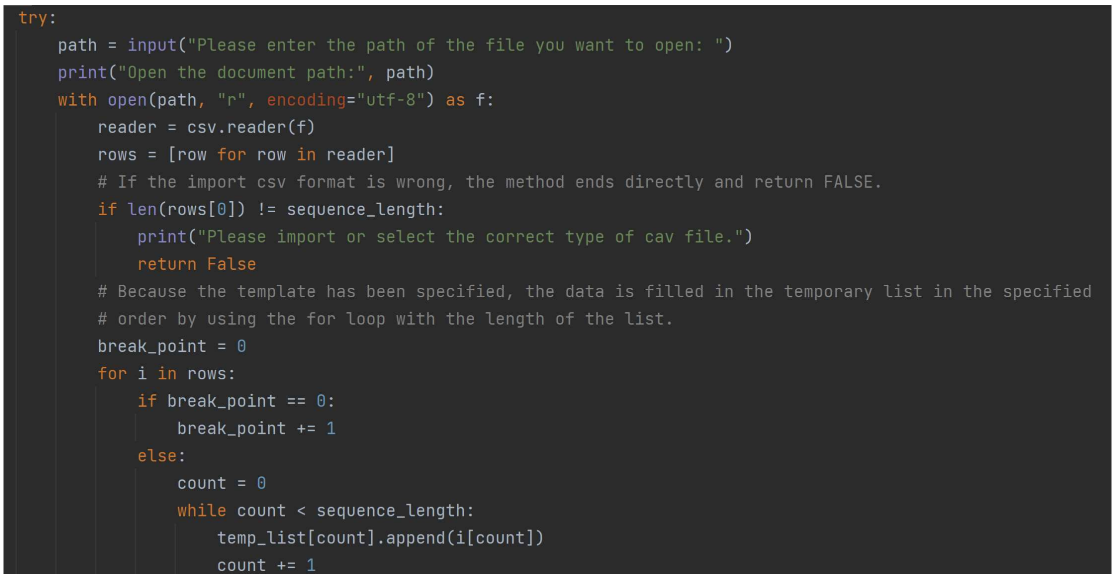
Code Implementation Components

```
configuration = [[], [], [], [], [], [], [], [], [], [], [], []]
sim_process = [[], [], [], []]
sim_limit = [[], [], [], [], [], [], [], []]
```

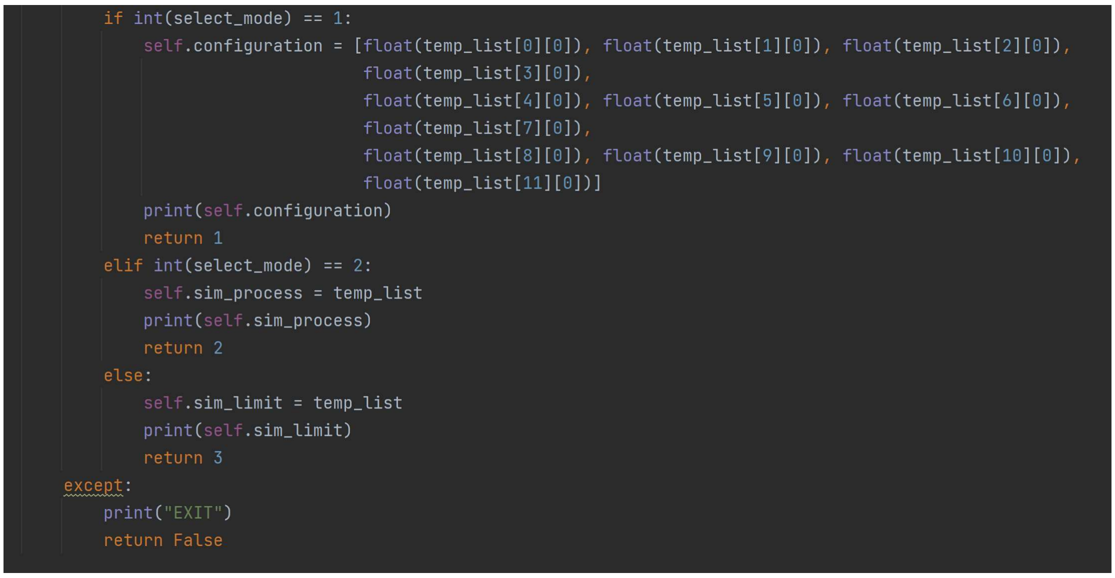
Standardize the format of the imported data. The standardized format can be classified for different csv and database formats. For example: It can be judged that the data should be imported into the table of the simulation process through the import list of length 4.



```python
    def input_csv(self):
        sequence_length = 0
        temp_list = []
        select_mode = input("Please select the type of file you: 1. Configuration; 2. Simulation process; 3. "
                            "Simulation limits: ")
        # Import the corresponding csv template through user options, and limit the length of the list.
        if int(select_mode) == 1:
            sequence_length = 12
            temp_list = self.configuration
        elif int(select_mode) == 2:
            sequence_length = 4
            temp_list = self.sim_process
        elif int(select_mode) == 3:
            sequence_length = 8
            temp_list = self.sim_limit

        else:
            print("Please enter the correct mode!")
            print("CLOSE!")
            return False
```



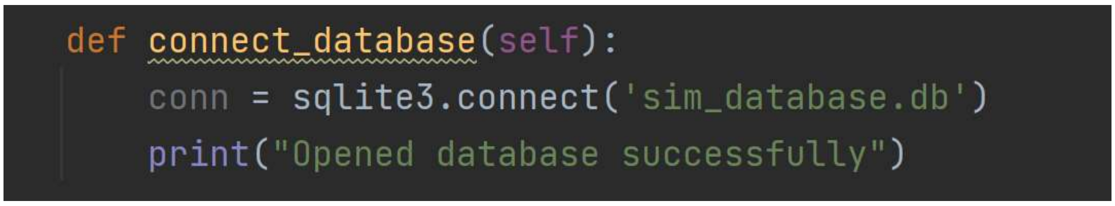
```python
    try:
        path = input("Please enter the path of the file you want to open: ")
        print("Open the document path:", path)
        with open(path, "r", encoding="utf-8") as f:
            reader = csv.reader(f)
            rows = [row for row in reader]
            # If the import csv format is wrong, the method ends directly and return FALSE.
            if len(rows[0]) != sequence_length:
                print("Please import or select the correct type of cav file.")
                return False
            # Because the template has been specified, the data is filled in the temporary list in the specified
            # order by using the for loop with the length of the list.
            break_point = 0
            for i in rows:
                if break_point == 0:
                    break_point += 1
                else:
                    count = 0
                    while count < sequence_length:
                        temp_list[count].append(i[count])
                        count += 1
```



```python
            if int(select_mode) == 1:
                self.configuration = [float(temp_list[0][0]), float(temp_list[1][0]), float(temp_list[2][0]),
                                      float(temp_list[3][0]),
                                      float(temp_list[4][0]), float(temp_list[5][0]), float(temp_list[6][0]),
                                      float(temp_list[7][0]),
                                      float(temp_list[8][0]), float(temp_list[9][0]), float(temp_list[10][0]),
                                      float(temp_list[11][0])]
                print(self.configuration)
                return 1
            elif int(select_mode) == 2:
                self.sim_process = temp_list
                print(self.sim_process)
                return 2
            else:
                self.sim_limit = temp_list
                print(self.sim_limit)
                return 3
        except:
            print("EXIT")
            return False
```

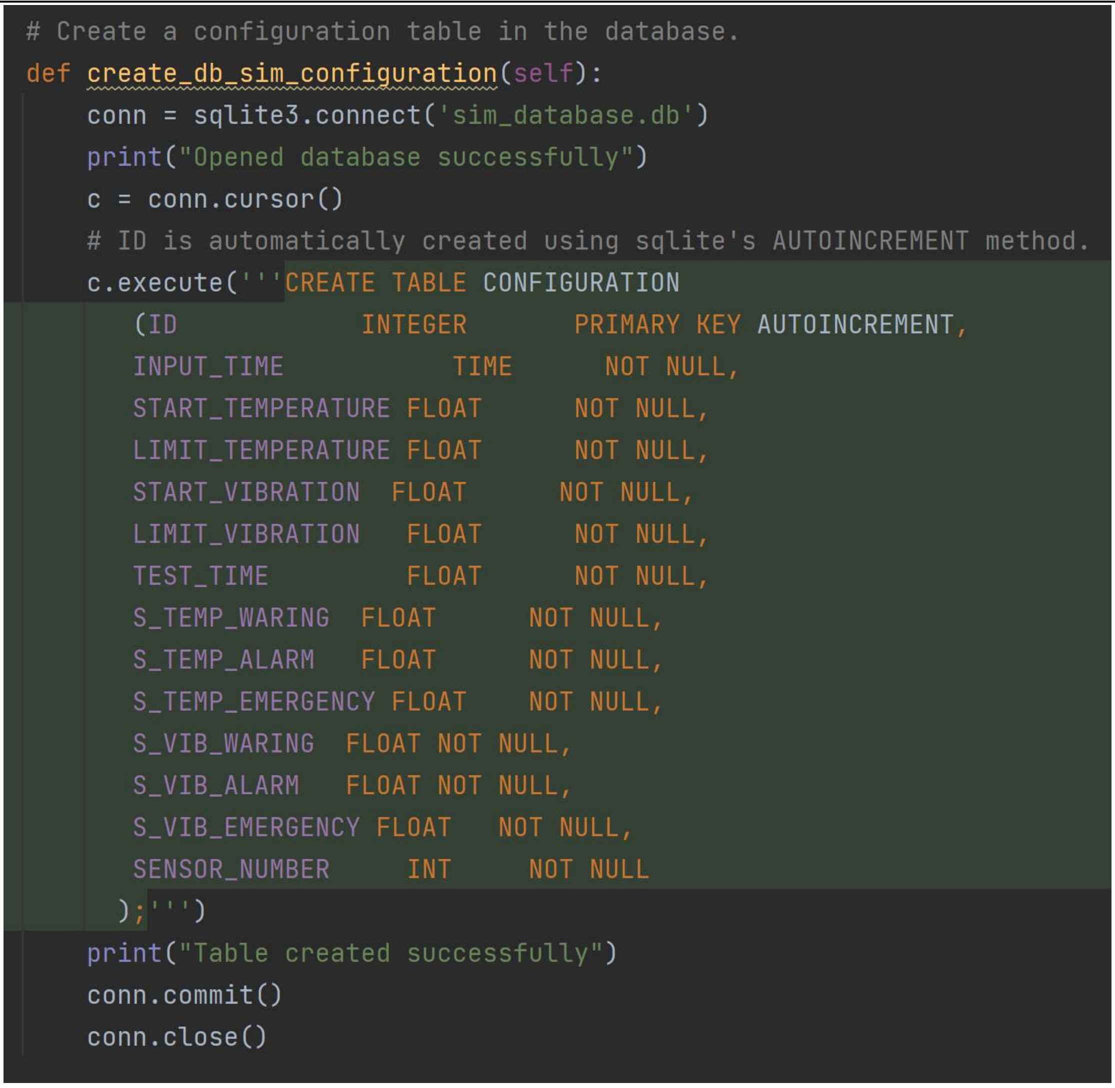
This class uses input() to guide the user to input the format specification of the imported data (csv).

The method uses the open and reader methods to read the csv of the local path.
By standardizing the length, the csv data is imported into the list in sequence in units of columns. And write to the class variable. The return value is FALSE (import failed) and the number 123 (representing the imported csv type).

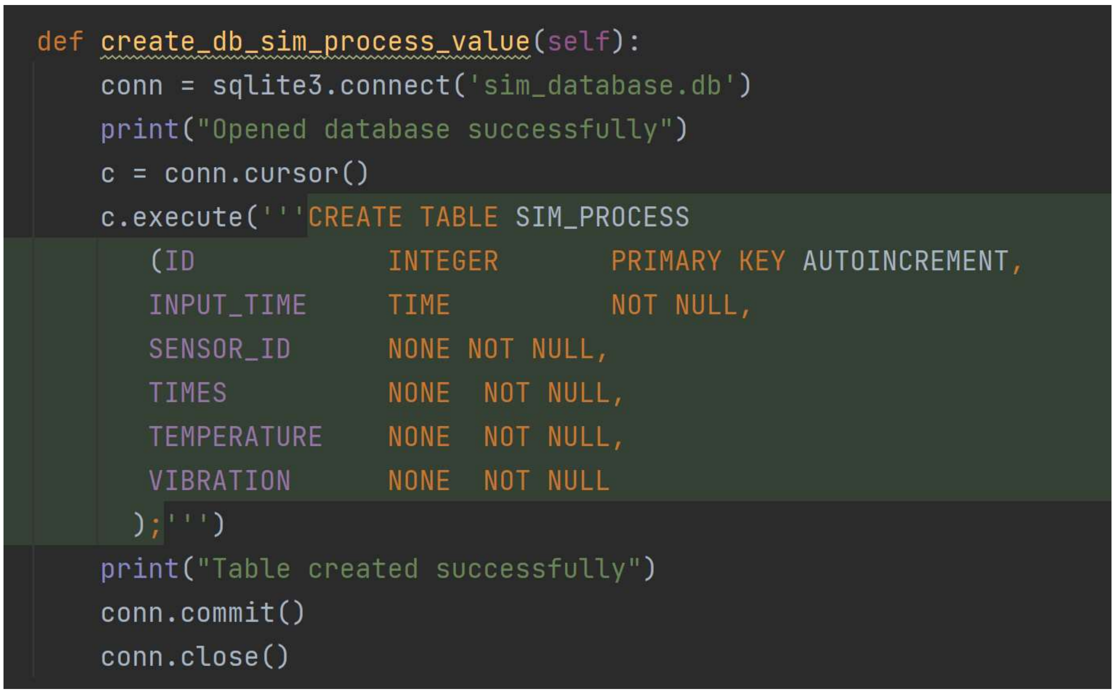


```python
def connect_database(self):
    conn = sqlite3.connect('sim_database.db')
    print("Opened database successfully")
```

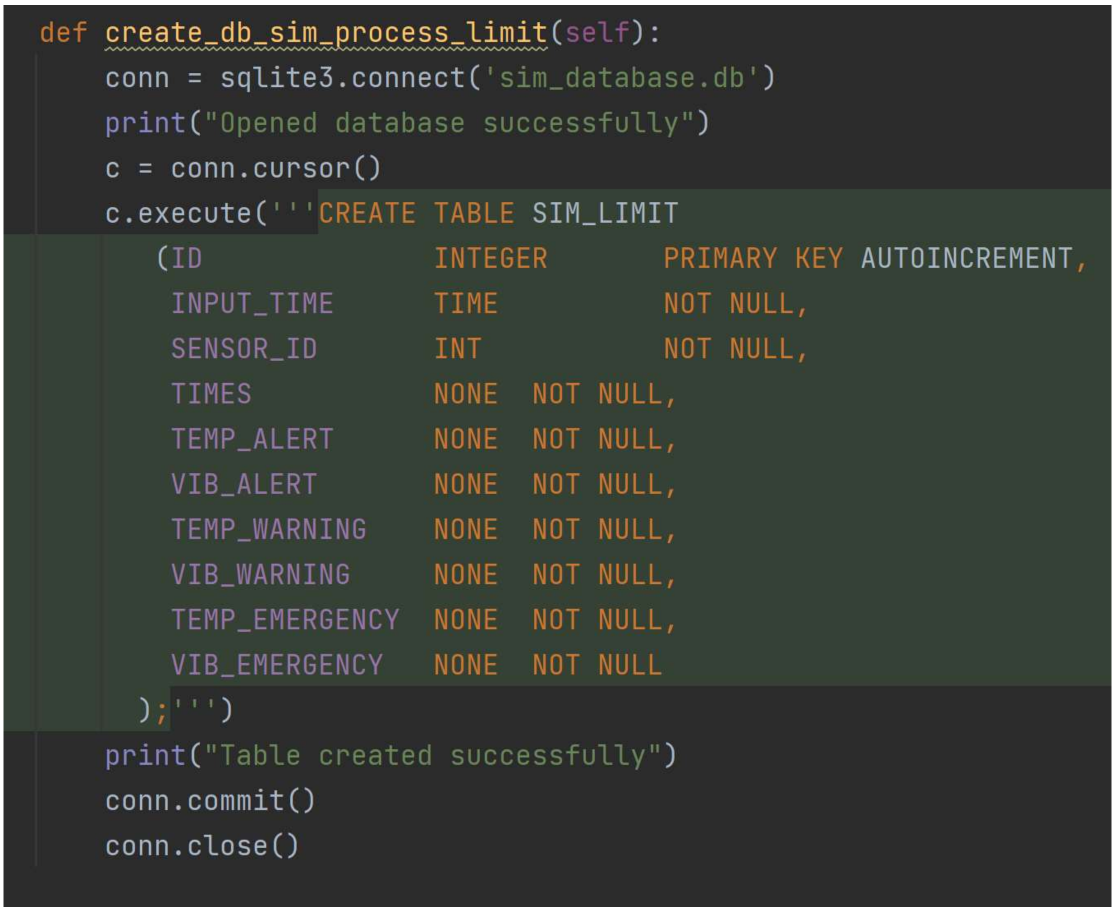
Sqlite3.connect() method can connect to the db file of the same name sqlite3 in the project directory. If there is no database with the same name, create a new database.



```python
# Create a configuration table in the database.
def create_db_sim_configuration(self):
    conn = sqlite3.connect('sim_database.db')
    print("Opened database successfully")
    c = conn.cursor()
    # ID is automatically created using sqlite's AUTOINCREMENT method.
    c.execute('''CREATE TABLE CONFIGURATION
       (ID            INTEGER       PRIMARY KEY AUTOINCREMENT,
       INPUT_TIME           TIME       NOT NULL,
       START_TEMPERATURE FLOAT      NOT NULL,
       LIMIT_TEMPERATURE FLOAT      NOT NULL,
       START_VIBRATION  FLOAT      NOT NULL,
       LIMIT_VIBRATION   FLOAT      NOT NULL,
       TEST_TIME         FLOAT      NOT NULL,
       S_TEMP_WARING  FLOAT      NOT NULL,
       S_TEMP_ALARM   FLOAT      NOT NULL,
       S_TEMP_EMERGENCY FLOAT    NOT NULL,
       S_VIB_WARING  FLOAT NOT NULL,
       S_VIB_ALARM   FLOAT NOT NULL,
       S_VIB_EMERGENCY FLOAT   NOT NULL,
       SENSOR_NUMBER     INT      NOT NULL
      );''')
    print("Table created successfully")
    conn.commit()
    conn.close()
```



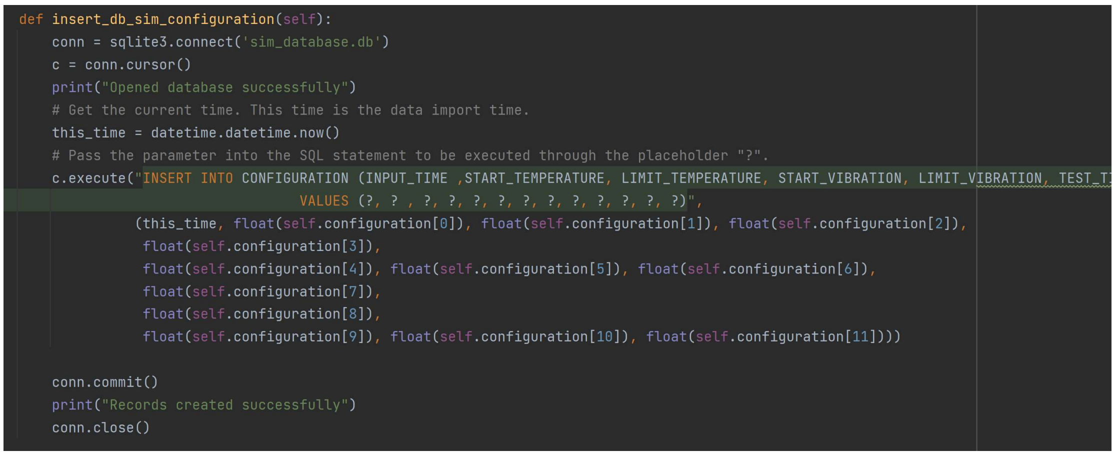
```
def create_db_sim_process_value(self):
    conn = sqlite3.connect('sim_database.db')
    print("Opened database successfully")
    c = conn.cursor()
    c.execute('''CREATE TABLE SIM_PROCESS
       (ID              INTEGER       PRIMARY KEY AUTOINCREMENT,
       INPUT_TIME       TIME          NOT NULL,
       SENSOR_ID        NONE NOT NULL,
       TIMES            NONE  NOT NULL,
       TEMPERATURE      NONE  NOT NULL,
       VIBRATION        NONE  NOT NULL
      );''')
    print("Table created successfully")
    conn.commit()
    conn.close()
```



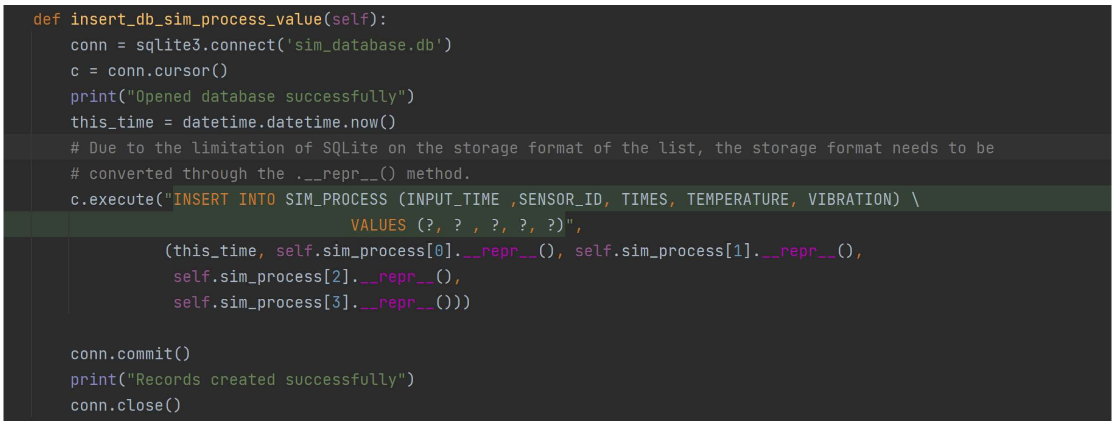
```
def create_db_sim_process_limit(self):
    conn = sqlite3.connect('sim_database.db')
    print("Opened database successfully")
    c = conn.cursor()
    c.execute('''CREATE TABLE SIM_LIMIT
       (ID              INTEGER       PRIMARY KEY AUTOINCREMENT,
        INPUT_TIME      TIME          NOT NULL,
        SENSOR_ID       INT           NOT NULL,
        TIMES           NONE  NOT NULL,
        TEMP_ALERT      NONE  NOT NULL,
        VIB_ALERT       NONE  NOT NULL,
        TEMP_WARNING    NONE  NOT NULL,
        VIB_WARNING     NONE  NOT NULL,
        TEMP_EMERGENCY  NONE  NOT NULL,
        VIB_EMERGENCY   NONE  NOT NULL
      );''')
    print("Table created successfully")
    conn.commit()
    conn.close()
```

The method is used to create a table of a specific template in the database for the user.
cursor() creates a cursor for this routine. After that, **execute** (sql [, optional parameters]) executes the statement in it through the cursor. Note that the statements inside are SQLite statements. Users need to use "?" and ":" placeholders to import parameters.
.commit() is used to commit the current transaction to make changes to the database.

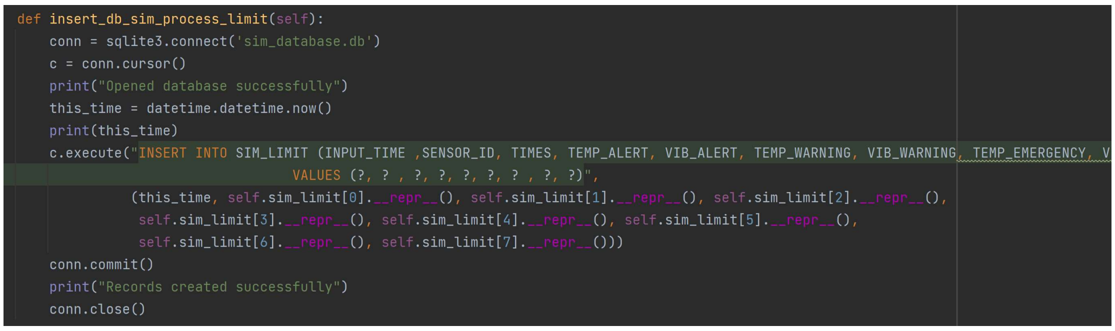
In addition, the parameter **AUTOINCREMENT** in SQLite can make the primary key in the table add ID to the data in order as new data is inserted.



```python
def insert_db_sim_configuration(self):
    conn = sqlite3.connect('sim_database.db')
    c = conn.cursor()
    print("Opened database successfully")
    # Get the current time. This time is the data import time.
    this_time = datetime.datetime.now()
    # Pass the parameter into the SQL statement to be executed through the placeholder "?".
    c.execute("INSERT INTO CONFIGURATION (INPUT_TIME ,START_TEMPERATURE, LIMIT_TEMPERATURE, START_VIBRATION, LIMIT_VIBRATION, TEST_TI
                                  VALUES (?, ? , ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
              (this_time, float(self.configuration[0]), float(self.configuration[1]), float(self.configuration[2]),
               float(self.configuration[3]),
               float(self.configuration[4]), float(self.configuration[5]), float(self.configuration[6]),
               float(self.configuration[7]),
               float(self.configuration[8]),
               float(self.configuration[9]), float(self.configuration[10]), float(self.configuration[11])))

    conn.commit()
    print("Records created successfully")
    conn.close()
```

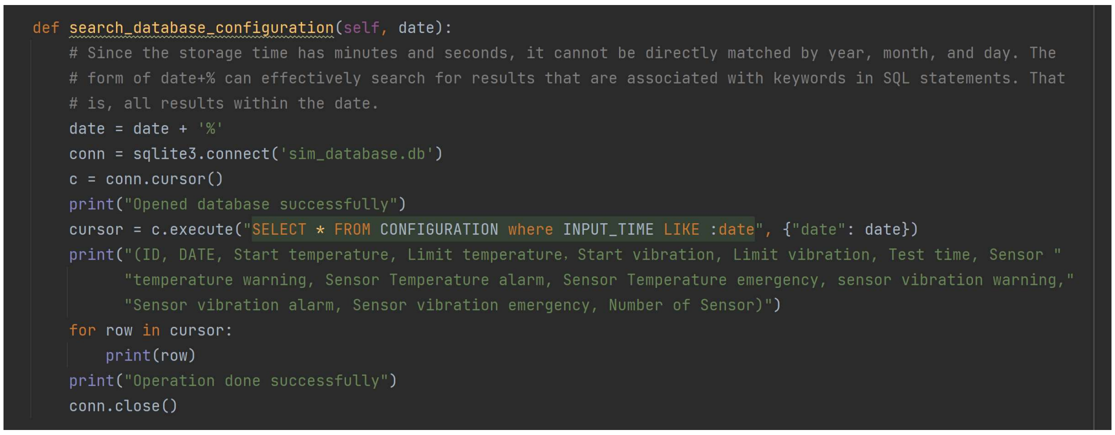


```python
def insert_db_sim_process_value(self):
    conn = sqlite3.connect('sim_database.db')
    c = conn.cursor()
    print("Opened database successfully")
    this_time = datetime.datetime.now()
    # Due to the limitation of SQLite on the storage format of the list, the storage format needs to be
    # converted through the .__repr__() method.
    c.execute("INSERT INTO SIM_PROCESS (INPUT_TIME ,SENSOR_ID, TIMES, TEMPERATURE, VIBRATION) \
                                  VALUES (?, ? , ?, ?, ?)",
              (this_time, self.sim_process[0].__repr__(), self.sim_process[1].__repr__(),
               self.sim_process[2].__repr__(),
               self.sim_process[3].__repr__()))

    conn.commit()
    print("Records created successfully")
    conn.close()
```

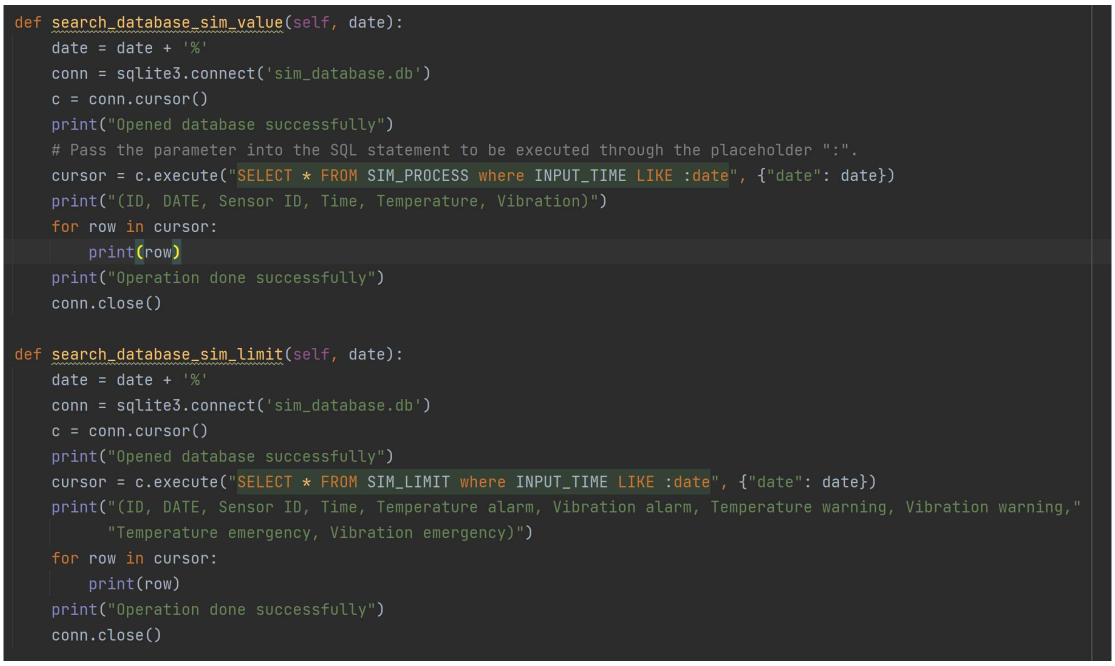


```python
def insert_db_sim_process_limit(self):
    conn = sqlite3.connect('sim_database.db')
    c = conn.cursor()
    print("Opened database successfully")
    this_time = datetime.datetime.now()
    print(this_time)
    c.execute("INSERT INTO SIM_LIMIT (INPUT_TIME ,SENSOR_ID, TIMES, TEMP_ALERT, VIB_ALERT, TEMP_WARNING, VIB_WARNING, TEMP_EMERGENCY, V
                                  VALUES (?, ? , ?, ?, ?, ?, ? , ?, ?)",
              (this_time, self.sim_limit[0].__repr__(), self.sim_limit[1].__repr__(), self.sim_limit[2].__repr__(),
               self.sim_limit[3].__repr__(), self.sim_limit[4].__repr__(), self.sim_limit[5].__repr__(),
               self.sim_limit[6].__repr__(), self.sim_limit[7].__repr__()))
    conn.commit()
    print("Records created successfully")
    conn.close()
```

The function of the insert method is to import the csv data variable in the standardized format into the database. For the list, you need to use the .__repr__() method to make it conform to the SQLite storage variable specification.

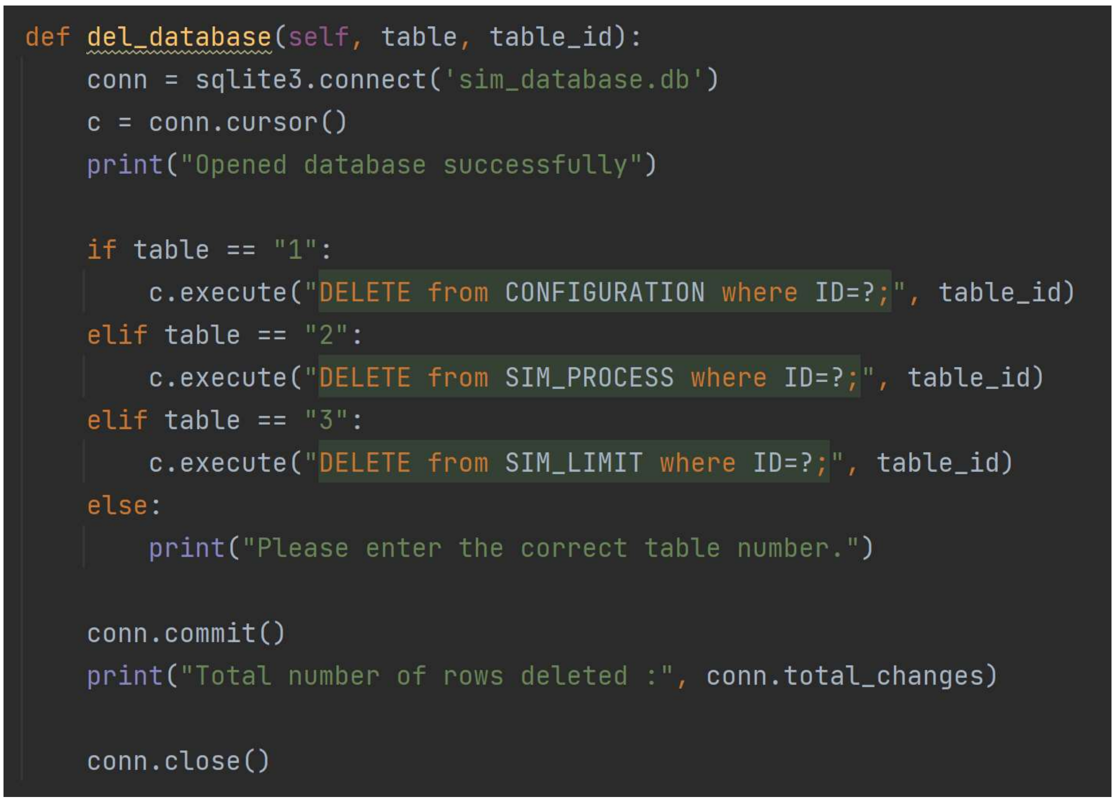


```python
def search_database_configuration(self, date):
    # Since the storage time has minutes and seconds, it cannot be directly matched by year, month, and day. The
    # form of date+% can effectively search for results that are associated with keywords in SQL statements. That
    # is, all results within the date.
    date = date + '%'
    conn = sqlite3.connect('sim_database.db')
    c = conn.cursor()
    print("Opened database successfully")
    cursor = c.execute("SELECT * FROM CONFIGURATION where INPUT_TIME LIKE :date", {"date": date})
    print("(ID, DATE, Start temperature, Limit temperature, Start vibration, Limit vibration, Test time, Sensor "
          "temperature warning, Sensor Temperature alarm, Sensor Temperature emergency, sensor vibration warning,"
          "Sensor vibration alarm, Sensor vibration emergency, Number of Sensor)")
    for row in cursor:
        print(row)
    print("Operation done successfully")
    conn.close()
```



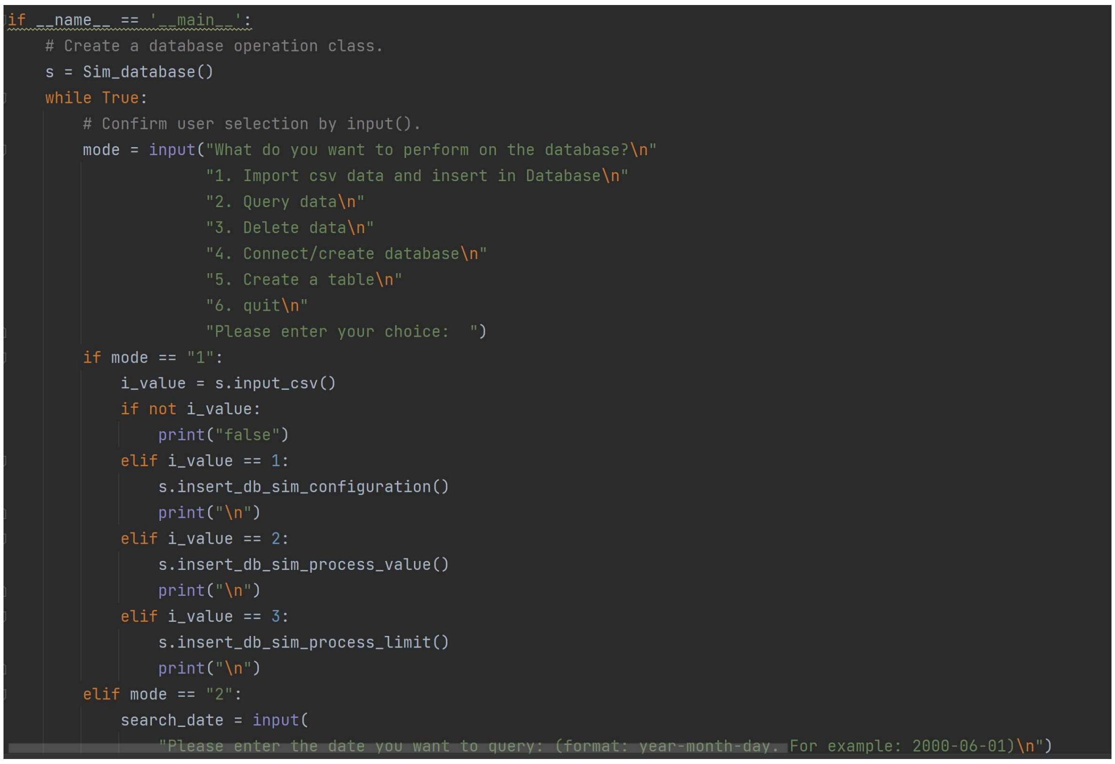
```python
def search_database_sim_value(self, date):
    date = date + '%'
    conn = sqlite3.connect('sim_database.db')
    c = conn.cursor()
    print("Opened database successfully")
    # Pass the parameter into the SQL statement to be executed through the placeholder ":".
    cursor = c.execute("SELECT * FROM SIM_PROCESS where INPUT_TIME LIKE :date", {"date": date})
    print("(ID, DATE, Sensor ID, Time, Temperature, Vibration)")
    for row in cursor:
        print(row)
    print("Operation done successfully")
    conn.close()

def search_database_sim_limit(self, date):
    date = date + '%'
    conn = sqlite3.connect('sim_database.db')
    c = conn.cursor()
    print("Opened database successfully")
    cursor = c.execute("SELECT * FROM SIM_LIMIT where INPUT_TIME LIKE :date", {"date": date})
    print("(ID, DATE, Sensor ID, Time, Temperature alarm, Vibration alarm, Temperature warning, Vibration warning,"
          "Temperature emergency, Vibration emergency)")
    for row in cursor:
        print(row)
    print("Operation done successfully")
    conn.close()
```

Where and like statements are used in the search method, the corresponding records in different tables are searched through the "year-month-day" entered by the user. The "value%" format simplifies user input. The user can ignore the minute and second record and then query it.

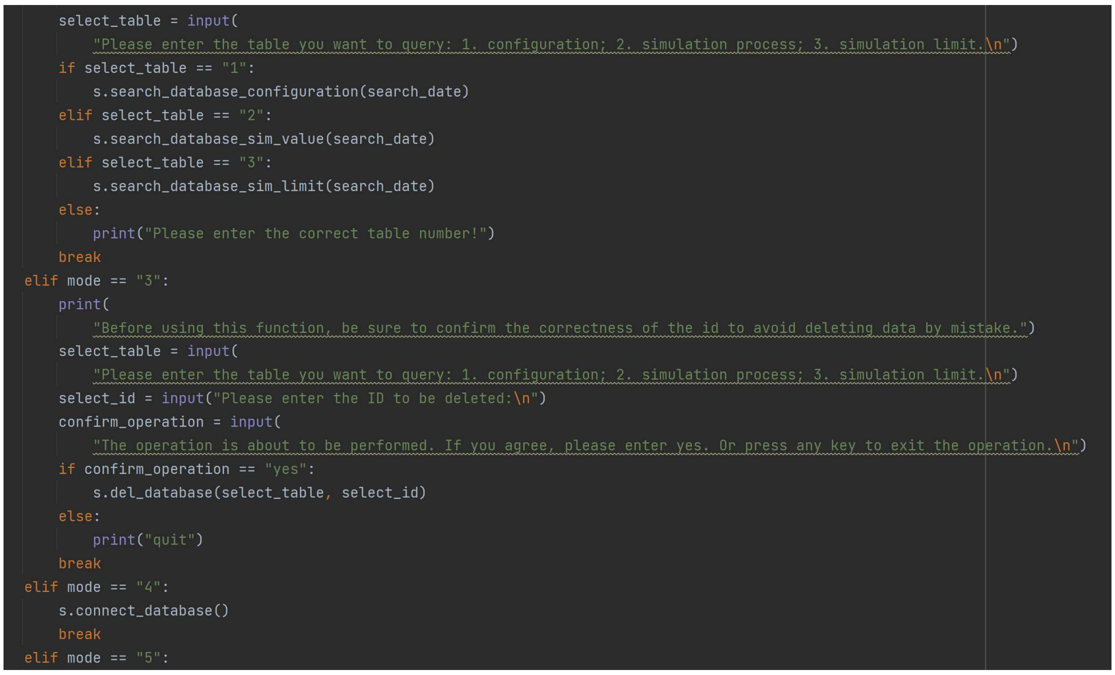


```python
def del_database(self, table, table_id):
    conn = sqlite3.connect('sim_database.db')
    c = conn.cursor()
    print("Opened database successfully")

    if table == "1":
        c.execute("DELETE from CONFIGURATION where ID=?;", table_id)
    elif table == "2":
        c.execute("DELETE from SIM_PROCESS where ID=?;", table_id)
    elif table == "3":
        c.execute("DELETE from SIM_LIMIT where ID=?;", table_id)
    else:
        print("Please enter the correct table number.")

    conn.commit()
    print("Total number of rows deleted :", conn.total_changes)

    conn.close()
```

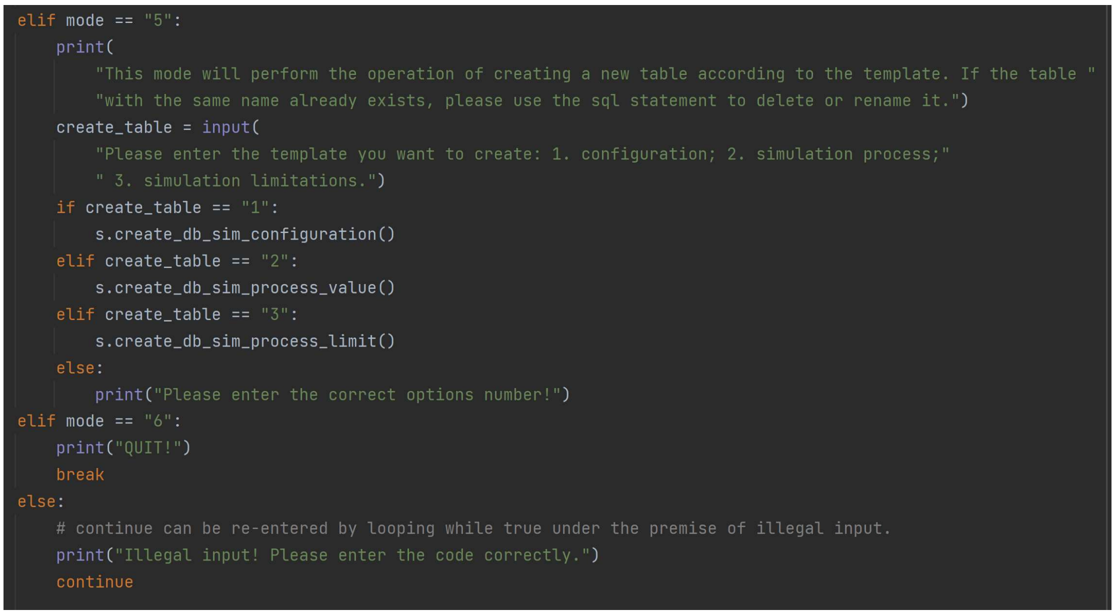
Through the id provided by the user, use DELETE to delete the data corresponding to the specific ID in the table.



```python
if __name__ == '__main__':
    # Create a database operation class.
    s = Sim_database()
    while True:
        # Confirm user selection by input().
        mode = input("What do you want to perform on the database?\n"
                     "1. Import csv data and insert in Database\n"
                     "2. Query data\n"
                     "3. Delete data\n"
                     "4. Connect/create database\n"
                     "5. Create a table\n"
                     "6. quit\n"
                     "Please enter your choice:  ")
        if mode == "1":
            i_value = s.input_csv()
            if not i_value:
                print("false")
            elif i_value == 1:
                s.insert_db_sim_configuration()
                print("\n")
            elif i_value == 2:
                s.insert_db_sim_process_value()
                print("\n")
            elif i_value == 3:
                s.insert_db_sim_process_limit()
                print("\n")
        elif mode == "2":
            search_date = input(
                "Please enter the date you want to query: (format: year-month-day. For example: 2000-06-01)\n")
```



```python
    select_table = input(
        "Please enter the table you want to query: 1. configuration; 2. simulation process; 3. simulation limit.\n")
    if select_table == "1":
        s.search_database_configuration(search_date)
    elif select_table == "2":
        s.search_database_sim_value(search_date)
    elif select_table == "3":
        s.search_database_sim_limit(search_date)
    else:
        print("Please enter the correct table number!")
    break
elif mode == "3":
    print(
        "Before using this function, be sure to confirm the correctness of the id to avoid deleting data by mistake.")
    select_table = input(
        "Please enter the table you want to query: 1. configuration; 2. simulation process; 3. simulation limit.\n")
    select_id = input("Please enter the ID to be deleted:\n")
    confirm_operation = input(
        "The operation is about to be performed. If you agree, please enter yes. Or press any key to exit the operation.\n")
    if confirm_operation == "yes":
        s.del_database(select_table, select_id)
    else:
        print("quit")
    break
elif mode == "4":
    s.connect_database()
    break
elif mode == "5":
```



```python
elif mode == "5":
    print(
        "This mode will perform the operation of creating a new table according to the template. If the table "
        "with the same name already exists, please use the sql statement to delete or rename it.")
    create_table = input(
        "Please enter the template you want to create: 1. configuration; 2. simulation process;"
        " 3. simulation limitations.")
    if create_table == "1":
        s.create_db_sim_configuration()
    elif create_table == "2":
        s.create_db_sim_process_value()
    elif create_table == "3":
        s.create_db_sim_process_limit()
    else:
        print("Please enter the correct options number!")
elif mode == "6":
    print("QUIT!")
    break
else:
    # continue can be re-entered by looping while true under the premise of illegal input.
    print("Illegal input! Please enter the code correctly.")
    continue
```

Use input() to get user input. Guide users to perform simple database operations through text reminders and if branches.