To write code that connects to a SQL database and performs operations on a table **inside a file-handling context**, you can use Python libraries like `sqlite3`, `mysql.connector`, or `psycopg2` (depending on the database). Here's an example for **SQLite**, which is a lightweight SQL database built into Python.

Example: Reading Data from a File and Inserting into SQL Database Table

Code to Connect and Work with SQL Database Inside File Handling

```
import sqlite3

# File handling and database connection
def process_file_and_insert_to_db(file_path, db_name):
    # Step 1: Open the file
    try:
        with open(file_path, 'r') as file:
            data = file.readlines()  # Read all lines from the
file
            # Step 2: Establish connection to the SQL database
            connection = sqlite3.connect(db_name)
            cursor = connection.cursor()

            # Step 3: Create a table (if it doesn't exist)
            cursor.execute('''
                CREATE TABLE IF NOT EXISTS records (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL,
                    age INTEGER NOT NULL,
                    score REAL NOT NULL
                )
            ''')
            print("Table created or already exists.")

            # Step 4: Insert data into the table
            for line in data:
                # Assuming each line in the file is formatted as
"name,age,score"
                name, age, score = line.strip().split(',')
                cursor.execute('''
                    INSERT INTO records (name, age, score)
                    VALUES (?, ?, ?)
                ''', (name, int(age), float(score)))

            # Step 5: Commit and close the database connection
            connection.commit()
            print("Data inserted successfully.")
    except FileNotFoundError:
        print(f"Error: The file '{file_path}' was not found.")
    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        # Step 6: Close the database connection
        if 'connection' in locals():
            connection.close()

# Example usage
file_path = 'data.txt'  # File containing "name,age,score" data
db_name = 'example.db'
process_file_and_insert_to_db(file_path, db_name)
```

Explanation:

1. **File Handling:**
    - Opens a file using `with open()`.
    - Reads each line of the file. Here, the file is assumed to contain comma-separated values (e.g., `John,25,89.5`).
2. **Database Connection:**
    - Connects to an SQLite database using `sqlite3.connect()`.
    - Creates a table (`CREATE TABLE IF NOT EXISTS`) if it doesn't already exist.

3. **Data Insertion:**
    - Each line from the file is split into `name`, `age`, and `score` using `split(',')`.
    - These values are inserted into the table using `cursor.execute()` with parameterized queries to prevent SQL injection.
4. **Error Handling:**
    - Captures potential errors such as `FileNotFoundError` for file issues or database errors.
5. **Cleanup:**
    - Ensures the database connection is closed in the `finally` block, whether or not an error occurs.

Example: Reading Data from SQL Table and Writing to a File

```
def export_db_to_file(db_name, output_file):
    try:
        # Step 1: Connect to the database
        connection = sqlite3.connect(db_name)
        cursor = connection.cursor()

        # Step 2: Fetch data from the table
        cursor.execute('SELECT * FROM records')
        rows = cursor.fetchall()

        # Step 3: Write the data to a file
        with open(output_file, 'w') as file:
            for row in rows:
                # Convert the row tuple to a comma-separated
string
                file.write(','.join(map(str, row)) + '\n')
        print(f"Data exported to '{output_file}' successfully.")
    except Exception as e:
        print(f"An error occurred: {e}")
    finally:
        # Step 4: Close the database connection
        if 'connection' in locals():
            connection.close()

# Example usage
export_db_to_file('example.db', 'output.txt')
```

Key Points:

- Use **file handling** to read from or write to files.
- Use a **database connection** inside the file-handling block or vice versa, depending on the workflow.
- Use parameterized queries (`?`) to safely insert data into the database.

To configure your code to connect to a specific database or table, you need to:

1. **Choose a database system** (e.g., SQLite, MySQL, PostgreSQL, etc.).
2. **Install the appropriate Python library** for interacting with the database.
3. **Provide connection details** (e.g., database name, host, username, password, etc.).
4. **Set up the database and table** if they don't already exist.

Step-by-Step Process for Configuring a Database/Table

Example 1: Using SQLite (No Configuration Required Beyond File Creation)

SQLite is a lightweight database system included with Python. The database is stored in a file on disk.

Code: Creating and Configuring SQLite Database and Table

```
import sqlite3

def configure_sqlite_database(db_name):
    # Connect to the SQLite database (creates the file if it
doesn't exist)
    connection = sqlite3.connect(db_name)
    cursor = connection.cursor()

    # Create a table if it doesn't already exist
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS employees (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            age INTEGER NOT NULL,
            department TEXT NOT NULL
        )
    ''')
    print(f"Database '{db_name}' and table configured
successfully.")

    # Commit changes and close the connection
    connection.commit()
    connection.close()

# Configure the SQLite database
db_name = 'company.db'
configure_sqlite_database(db_name)
```

How to Use:

- Run the code to create the database file (`company.db`) and a table named `employees`.
- Insert or query data using this table.

Example 2: Using MySQL

For MySQL, you need to:

Install MySQL Connector: Use the `mysql-connector-python` library.

```
pip install mysql-connector-python
```

1.
2. **Provide Connection Details**: Include host, user, password, and database name.

Code: Connecting to MySQL and Configuring a Table

```
import mysql.connector

def configure_mysql_database(host, user, password, database):
    try:
        # Connect to the MySQL server
        connection = mysql.connector.connect(
            host=host,
            user=user,
            password=password
        )
        cursor = connection.cursor()

        # Create a database if it doesn't exist
        cursor.execute(f"CREATE DATABASE IF NOT EXISTS
{database}")
        print(f"Database '{database}' created or already
exists.")
```

```
        # Connect to the specific database
        connection.database = database

        # Create a table if it doesn't exist
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS employees (
                id INT AUTO_INCREMENT PRIMARY KEY,
                name VARCHAR(255) NOT NULL,
                age INT NOT NULL,
                department VARCHAR(255) NOT NULL
            )
        ''')
        print("Table 'employees' configured successfully.")

        # Close the connection
        cursor.close()
        connection.close()
    except mysql.connector.Error as e:
        print(f"Error: {e}")

# Configure the MySQL database
configure_mysql_database(
    host='localhost',
    user='root',
    password='yourpassword',
    database='company_db'
)
```

How to Use:

- Replace `localhost`, `root`, `yourpassword`, and `company_db` with your MySQL server details.
- Run the code to create the `company_db` database and configure the `employees` table.

Example 3: Using PostgreSQL

For PostgreSQL, you need:

Install the `psycopg2` library:

```
pip install psycopg2
```

1.
2. **Provide Connection Details**.

Code: Connecting to PostgreSQL and Configuring a Table

```
import psycopg2

def configure_postgresql_database(host, user, password,
database):
    try:
        # Connect to the PostgreSQL server
        connection = psycopg2.connect(
            host=host,
            user=user,
            password=password,
            dbname=database
        )
        cursor = connection.cursor()

        # Create a table if it doesn't exist
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS employees (
                id SERIAL PRIMARY KEY,
                name VARCHAR(255) NOT NULL,
                age INT NOT NULL,
                department VARCHAR(255) NOT NULL
            )
        ''')
        print("Table 'employees' configured successfully.")

        # Commit changes and close the connection
        connection.commit()
        cursor.close()
        connection.close()
    except psycopg2.Error as e:
        print(f"Error: {e}")

# Configure the PostgreSQL database
configure_postgresql_database(
    host='localhost',
    user='postgres',
    password='yourpassword',
    database='company_db'
)
```

How to Use:

- Replace `localhost`, `postgres`, `yourpassword`, and `company_db` with your PostgreSQL server details.
- Run the code to create the `employees` table.

Integrating File Handling with Database Configuration

You can combine file-handling code with any of these examples. For instance:

```
def insert_data_from_file_to_db(file_path, db_name):
    try:
        # Open the file
        with open(file_path, 'r') as file:
            data = file.readlines()

        # Connect to the SQLite database
        connection = sqlite3.connect(db_name)
        cursor = connection.cursor()

        # Insert data into the table
        for line in data:
            name, age, department = line.strip().split(',')
            cursor.execute('''
                INSERT INTO employees (name, age, department)
                VALUES (?, ?, ?)
            ''', (name, int(age), department))

        # Commit changes
        connection.commit()
        print("Data inserted successfully.")
    except Exception as e:
        print(f"Error: {e}")
    finally:
        # Close the database connection
        if 'connection' in locals():
            connection.close()

# Example usage
insert_data_from_file_to_db('employees.txt', 'company.db')
```