In the RestaurantDatabase.py which is responsible for managing the connection to a MySQL database and performing various operations on the database tables related to a restaurant reservation system.

1. Class Definition:-

The RestaurantDatabase class is defined with an '__init__' method that initializes the class instance with the necessary parameters for establishing a database connection.
The '__init__' method takes the following parameters: 'host', 'port', 'database', 'user', and 'password'. These parameters are used to configure the database connection settings.
Inside the __init__ method, the provided parameters are assigned to the corresponding instance attributes (self.host, self.port, self.database, self.user, self.password).
The self.connection and self.cursor attributes are initialized to None. Finally, the _connect_() method is called to establish the database connection.

2. Database Connection:-

The _connect_() method is responsible for establishing a connection to the MySQL database. It uses the mysql.connector.connect() function to create a connection object, passing the necessary connection parameters (host, port, database, user, password) from the class instance. If the connection is successful, a message "Successfully connected to the database" is printed. If an error occurs during the connection process, the except block catches the Error exception and prints an error message.

3. Database Operations:-

The class defines several methods to perform various operations on the database tables.

a. _addReservation_(self, customer_id, reservation_time, number_of_guests, special_requests):-

- This method inserts a new reservation into the reservations table.
- It first checks if the database connection is established (self.connection.is_connected()).
- If the connection is established, it creates a cursor object (self.cursor = self.connection.cursor()).
- The SQL INSERT query is constructed to insert the provided values into the reservations table.
- The query is executed using the self.cursor.execute() method, passing the SQL query and the values as parameters.
- The changes are committed to the database using self.connection.commit().
- A success message is printed.

b. _getAllReservations_(self):-

- This method retrieves all reservations from the `reservations` table.
- It first checks if the database connection is established.

- If the connection is established, it creates a cursor object.
- The SQL `SELECT` query is constructed to retrieve all records from the `reservations` table.
- The query is executed using `self.cursor.execute()`.
- The result records are fetched using `self.cursor.fetchall()` and returned.

c. _addCustomer_(self, customer_name, contact_info):-

- This method inserts a new customer into the customers table.
- It first checks if the database connection is established.
- If the connection is established, it creates a cursor object.
- The customer details (customer_name and contact_info) are printed for debugging purposes.
- The SQL INSERT query is constructed to insert the provided values into the customers table.
- The query is executed using self.cursor.execute(), passing the SQL query and the values as parameters.
- The changes are committed to the database using self.connection.commit().
- A success message is printed.

d. _getCustomerPreferences_(self, customer_id):-

- This method retrieves the dining preferences for a specific customer from the diningPreferences table.
- It first checks if the database connection is established.
- If the connection is established, it creates a cursor object.
- The SQL SELECT query is constructed to retrieve records from the diningPreferences table where the customerId matches the provided customer_id.
- The query is executed using self.cursor.execute(), passing the SQL query and the customer_id as a parameter.
- The result records (preferences) are fetched using self.cursor.fetchall() and returned.

The **RestaurantDatabase** class provides a convenient way to interact with the restaurant reservation database by encapsulating the database connection and SQL operations within its methods. Developers can create an instance of this class and use its methods to perform various tasks related to reservations, customers, and dining preferences.