The required task is to build a generic parallel sort and parallel join algorithm.

1. Implement a Python function **ParallelSort()** that takes as input: (1) **InputTable** stored in a PostgreSQL database, (2) **SortingColumnName** the name of the column used to order the tuples by. **ParallelSort()** then sorts all tuples (using five parallelized threads) and stores the sorted tuples for in a table named **OutputTable** (the output table name is passed to the function). The **OutputTable** contains all the tuple present in **InputTable** sorted in ascending order.

   Function Interface: -
   ParallelSort (InputTable, SortingColumnName, OutputTable, openconnection)
   InputTable – Name of the table on which sorting needs to be done.
   SortingColumnName – Name of the column on which sorting needs to be done, would be either of type integer or real or float. Basically Numeric format. Will be Sorted in Ascending order.
   OutputTable – Name of the table where the output needs to be stored.
   openconnection – connection to the database.

2. Implement a Python function **ParallelJoin()** that takes as input: (1) **InputTable1** and **InputTable2** table stored in a PostgreSQL database, (2) **Table1JoinColumn** and **Table2JoinColumn** that represent the join key in each input table respectively. **ParallelJoin()** then joins both **InputTable1** and **InputTable2** (using five parallelized threads) and stored the resulting joined tuples in a table named **OutputTable** (the output table name is passed to the function). The schema of **OutputTable** should be similar to the schema of both **InputTable1** and **InputTable2** combined.

   Function Interface: -
   ParallelJoin (InputTable1, InputTable2, Table1JoinColumn, Table2JoinColumn, OutputTable, openconnection)
   InputTable1 – Name of the first table on which you need to perform join.
   InputTable2 – Name of the second table on which you need to perform join.
   Table1JoinColumn – Name of the column from first table i.e. join key for first table.
   Table2JoinColumn – Name of the column from second table i.e. join key for second table.
   OutputTable - Name of the table where the output needs to be stored.
   openconnection – connection to the database.

Please use the function signature exactly same as mentioned in Assignment3_Interface.py.
You will notice that in top of Assignment3_Interface.py, following things are declared
FIRST_TABLE_NAME = 'table1'
SECOND_TABLE_NAME = 'table2'
SORT_COLUMN_NAME_FIRST_TABLE = 'column1'
SORT_COLUMN_NAME_SECOND_TABLE = 'column2'
JOIN_COLUMN_NAME_FIRST_TABLE = 'column1'
JOIN_COLUMN_NAME_SECOND_TABLE = 'column2'

Let's use an example to understand these fields and their usage.
Example: - So, once your implementation is complete, you will have to create two tables in database manually lets name them *MovieRating* and *MovieBoxOfficeCollection*. Suppose, you

want to sort MovieRating by column *Rating* and *MovieBoxOfficeCollection* by column *Collection*. You also want to join *MovieRating* and *MovieBoxOfficeCollection* by column *MovieID*. Then, you would define the variables mentioned above as :
FIRST_TABLE_NAME = '*MovieRating'*
SECOND_TABLE_NAME = '*MovieBoxOfficeCollection'*
SORT_COLUMN_NAME_FIRST_TABLE = '*Rating'*
SORT_COLUMN_NAME_SECOND_TABLE = '*Collection'*
JOIN_COLUMN_NAME_FIRST_TABLE = *'MovieID'*
JOIN_COLUMN_NAME_SECOND_TABLE = '*MovieID'*

Instructions on how this will be tested: - Please follow these instructions closely.

1. Two tables would be created in the database manually.
2. The created tables would contain at least an integer field, which would be used for both Parallel Sorting and Parallel Joining.
3. Then, the ParallelSort() and ParallelJoin() Function would be called to check the correctness of the program.
4. Your code should use 5 threads for both ParallelSort() as well as ParallelJoin().
5. Your code should be able to handle table irrespective of its schema.
6. Do not make your code dependent on any particular table; it should be able to work on any table.