

DSA8002 Assignment

Course Work

Anjum Banu Ismail

40313865

aismail03@qub.ac.uk

Contents

[1. Objective 2](#_Toc66214738)

[2. Decision and Assumptions 2](#_Toc66214739)

[3. Software and Installations 2](#_Toc66214740)

[4. Functional Behaviour 3](#_Toc66214741)

[5. Technical Documentation 4](#_Toc66214742)

[5.1 Project Directory Structure 4](#_Toc66214743)

[5.2 Code Engine and Data base creation 5](#_Toc66214744)

[5.3 Normalisation and Database operations 7](#_Toc66214745)

[5.4 User operations and executions 10](#_Toc66214746)

[5.5 Quick look at console 11](#_Toc66214747)

[5.6 Unit Testing 12](#_Toc66214748)

[6. Source Code 12](#_Toc66214749)

[6.1 Main.py 12](#_Toc66214750)

[6.2 Package process 12](#_Toc66214751)

[6.2.1 listingConstants.py 12](#_Toc66214752)

[6.2.2 queryConstant.py 13](#_Toc66214753)

[6.2.3 listingUtil.py 14](#_Toc66214754)

[6.3 Package "database” 15](#_Toc66214755)

[6.3.1 database\_operations.py 15](#_Toc66214756)

[6.3.2 crud\_operations.py 15](#_Toc66214757)

[6.3.3 create.py 16](#_Toc66214758)

[6.3.4 insert.py 16](#_Toc66214759)

[6.3.5 select.py 17](#_Toc66214760)

[6.3.6 update.py 17](#_Toc66214761)

[6.3.7 delete.py 17](#_Toc66214762)

[7.1 Package "tables” 18](#_Toc66214763)

[7.1.1 id.py 18](#_Toc66214764)

[7.1.2 name.py 18](#_Toc66214765)

[7.1.3 room.py 18](#_Toc66214766)

[7.1.4 host.py 19](#_Toc66214767)

[7.1.5 room\_info.py 19](#_Toc66214768)

[7.1.6 room\_reviews.py 20](#_Toc66214769)

[7.1.7 room\_location.py 20](#_Toc66214770)

[7.1.8 room\_host\_mapping.py 21](#_Toc66214771)

[7.1.9 neighbourhood\_region.py 21](#_Toc66214772)

[8. UML Diagram 22](#_Toc66214773)

[9. Test Code 22](#_Toc66214774)

[9.1 test\_process.py 22](#_Toc66214775)

[9.2 test\_table.py 23](#_Toc66214776)

[9.3 test\_database.py 26](#_Toc66214777)

[9.4 test\_complete.py 26](#_Toc66214778)

[10. Executions 27](#_Toc66214779)

[11. References 27](#_Toc66214780)

## Objective

The main objective of this Python project is to implement an application which takes CSV file an input and generates a normalised data by updating the entries into the data base.

This project will also support the functionality to retrieve the results from data base based one query provided.

## Decision and Assumptions

* This Python project is implemented to normalise only the “listings.csv” data downloaded from kaggle.

<https://www.kaggle.com/jojoker/singapore-airbnb>

* All the required software and libraries are installed before running this application. [(Please refer section 3)](#_Decision_and_Assumptions)
* Anyone using this application need not download the csv file, This is already done and placed in “csvData” folder under the root directory of the project
* The newly created data base and the retrieved data from database is placed in “output” folder under the root directory of the project
* Data is stored with new column names for better understanding.

## Software and Installations

* **Python (Latest version 3.9)-** <https://www.python.org/downloads/>

Once installed verify the python version in cmd (windows) or Terminal (MAC)

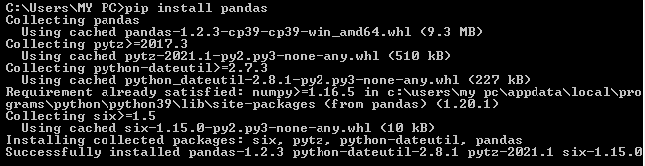


**Note:** For Mac users the command is **python3 –version**

* **Python libraries – (numpy and pandas)**

These libraries can be install from command line by using pip package manager. Pip downloads the package and notifies you if it is successfully installed.





* **Pycharm IDE (Community Edition)** <https://www.jetbrains.com/pycharm/download>
* **DB Browser for SQLite** <https://sqlitebrowser.org/dl>

## Functional Behaviour

Below are functional implementation handled as part of this application

**Step 1: Load the csv file:**

The required "**listings.csv**" is already loaded in "**csvData**" directory , the raw data from csv file is pulled and converted in data frame.

**Step 2: Normalisation of Data:**

The current data frame has raw data and hence normalisation is performed to create multiple data frames with proper primary and foreign key mapping.

**Step 3: Creating a new data base and creating tables**

Once the tables are created and loaded with data, success message is shown on console.

**Step 4: Cleaning the Data:**

The Normalised data is now cleaned by:

* Removing duplicate entries
* Adding default values for NULL entries only for certain entries.

**Step 5: Asking user inputs to perform CRUD operations**

User is asked to choose one of the options to INSERT / UPDATE / DELETE / SELECT

User is also asked to enter a query to be executed

**Step 6: Execution of Query given**

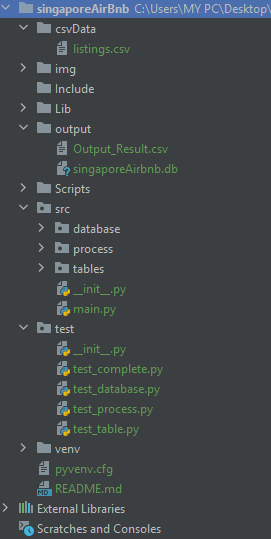
The query given in console is executed and in case if SELECT operation the first 5 rows of the output result is displayed and then the entire data is copied to **output\_Result.csv** in "**output**" directory

## Technical Documentation

## Project Directory Structure

* The project name with a root directory named “**singaporeAirBnb**” and below is an overview of sub directories created within the project:

Raw Csv file is placed here



The output of select query is updated in excel sheet

New Database is created every time when programm is executed

All the operation are performed within this src folder and main.py should be used to execute the programme

All the test cases created are placed within the test folder. Test\_complete.py has a test suite created to run all the test case at once

* The project should be executed using the “main.py” file within src folder.

File: main.py

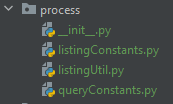
Path: src/main.py

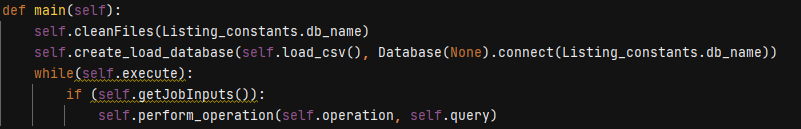
from src.process.listingUtil import ListingUtil  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 listing = ListingUtil()  
 listing.main()

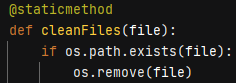
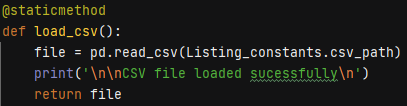
## Code Engine and Data base creation

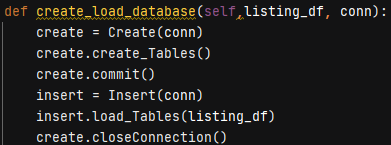
On execution of main file function main is called in **listingUtil** package under **process** directory which is used to maintain the operations by calling the required classes

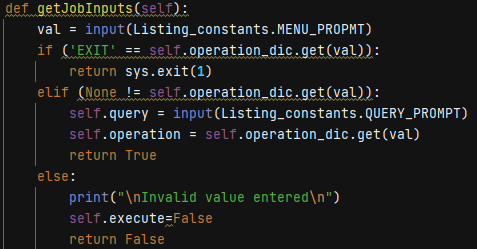
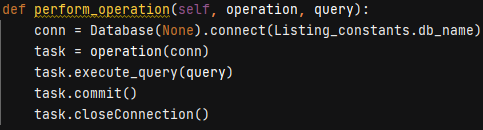
This is the heart of the project where all the 6 steps mentioned under functional Behaviour are executed





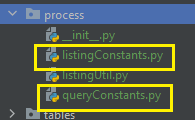
 



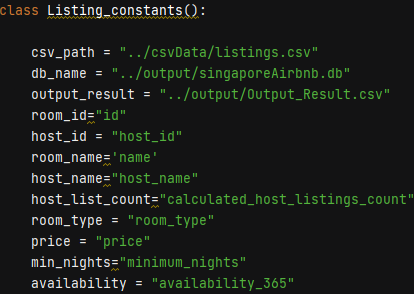
 

This particular code will perform the below activities

* Deletion of any existing db file if already exist in the path by calling cleanFiles(db\_name)
* Load the raw csv from the given path by calling load\_csv()
* Make a data base connection by providing the db name connect(db\_name)
* The Data base is created with name “**SingaporeAirBnb.db**”
* The process directory also includes 2 constant files, where the frequently used variables name is maintained in one single file to avoid any ambiguity and typo errors.



**listingConstants.py** mostly maintains the variable names for the raw data set column, file path etc, below is the glimpse



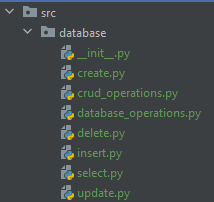
Any modification in path for storing or retrieving the data should be done here

On the other hand **queryConstant.py** maintains the “CREATE TABLE” queries for the all the tables getting created, any new table to be added in future the query can be updated here and can be used in their respective table class

|  |
| --- |
| class Query\_Constants:   create\_room= "CREATE TABLE ROOM (ID INTEGER PRIMARY KEY NOT NULL UNIQUE, " \  "NAME VARCHAR(30) NOT NULL)"   create\_host= "CREATE TABLE HOST (ID INTEGER PRIMARY KEY NOT NULL UNIQUE, " \  "NAME VARCHAR(30) NOT NULL," \  "LISTING\_COUNT INTEGER NOT NULL)"   create\_room\_reviews= "CREATE TABLE ROOM\_REVIEWS(ROOM\_ID INTEGER NOT NULL, " \  "NO\_OF\_REVIEWS INTEGER NOT NULL, " \  "LAST\_REVIEW DATE NOT NULL," \  "REVIEW\_PER\_MONTH FLOAT NOT NULL, " \  "FOREIGN KEY (ROOM\_ID) REFERENCES ROOM(ID))"   create\_room\_info= "CREATE TABLE ROOM\_INFO(ROOM\_ID INTEGER NOT NULL," \  "ROOM\_TYPE VARCHAR(30) NOT NULL, " \  "ROOM\_PRICE FLOAT NOT NULL," \  "MINIMUM\_NIGHTS INTEGER NOT NULL," \  "AVAILABILITY INTEGER NOT NULL," \  "FOREIGN KEY (ROOM\_ID) REFERENCES ROOM(ID))"   create\_room\_location= "CREATE TABLE ROOM\_LOCATION(ROOM\_ID INTEGER NOT NULL, " \  "LATITUDE Float NOT NULL, " \  "LONGITUDE VARCHAR(30) NOT NULL, " \  "NEIGHBOURHOOD VARCHAR(30) NOT NULL, " \  "FOREIGN KEY (NEIGHBOURHOOD) REFERENCES NEIGHBOURHOOD\_REGION(NEIGHBOURHOOD)," \  "FOREIGN KEY (ROOM\_ID) REFERENCES ROOM(ID))"    create\_room\_host\_mapping= "CREATE TABLE ROOM\_HOST\_MAPPING(ROOM\_ID INTEGER NOT NULL UNIQUE," \  "HOST\_ID INTEGER NOT NULL," \  "FOREIGN KEY (ROOM\_ID) REFERENCES ROOM(ID)," \  "FOREIGN KEY (HOST\_ID) REFERENCES HOST(ID))"    create\_neighbourhood\_region = "CREATE TABLE NEIGHBOURHOOD\_REGION(NEIGHBOURHOOD VARCHAR(30) PRIMARY KEY NOT NULL UNIQUE, " \  "REGION VARCHAR(30) NOT NULL)" |

## Normalisation and Database operations

The execution is these create statement is carried out in **crud\_operations.py** under database directory.



There is a multilevel inheritance created between these classes and below is the structure and behaviour of each class

|  |  |  |  |
| --- | --- | --- | --- |
| **Class name** | **Parent class name** | **Comments** | **Task performed** |
| database\_operation.py |  | All the task related to data base is performed here | Connecting data base  Commit  Rollback  Close connection |
| crud\_operation.py | database\_operation.py | All the query execution tasks are performed here | Execute query |
| select.py | crud\_operation.py | Inherits the methods from parent class and overrides some methods based on operations | Inherits the methods from parent class |
| insert.py | crud\_operation.py |
| delete.py | crud\_operation.py |
| update.py | crud\_operation.py |

Any of the CRUD operations are performed, then the connection to database and commits are inherited from its parent classes.

* The data is normalised into 7 tables and below is entity relationship diagram for the same.

**Actual raw data columns:**

id

name

host\_id

host\_name

neighbourhood\_group

neighbourhood

latitude

longitude

room\_type

price

minimum\_nights

number\_of\_reviews

last\_review

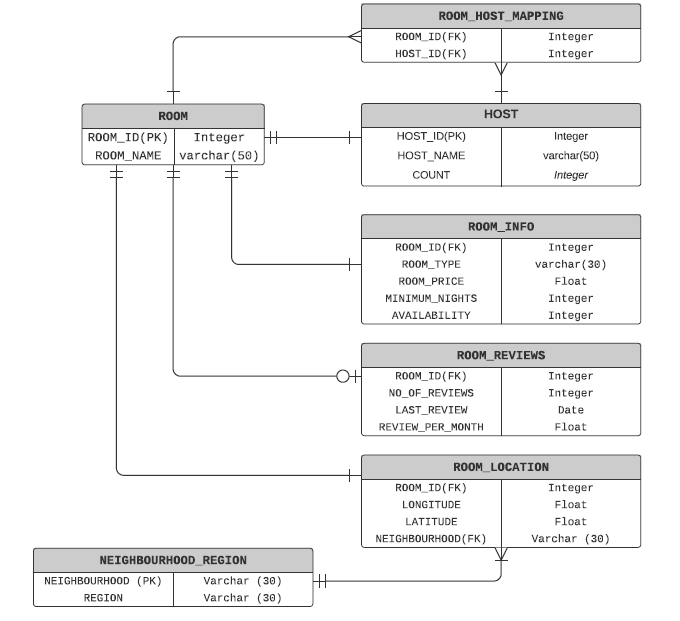
reviews\_per\_month

calculated\_host\_listings\_count

availability\_365

**Normalised data ER diagram:**

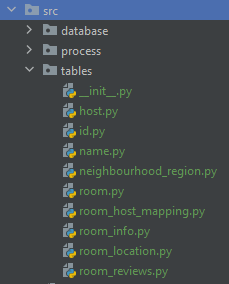
1NF, 2NF and 3NF is applied for the data



Once we are confident on the normalisation strategy, the code from Directory tables is pulled to get the required data to be loaded to data base.

The code also performs any modification required by removing duplicate data and also updating Nan records. All the required operation for inserting data in tables is performed by below code.

There is separate class created for every table where “**id.py**” and “**name.py**” are parent class and rest all are child classes which inherits the parent classes for primary key and common data cleaning methods



* Once the data is normalised and loaded in data based by performing the cleaning activities the data size is reduced and that can be proved with the below table

Total entries in the raw data: 7907

**After normalisation:**

|  |  |
| --- | --- |
| **Tables** | **No of rows** |
| ROOM | 7907 |
| HOST | 2705 |
| ROOM\_HOST\_MAPPING | 7907 |
| ROOM\_INFO | 7907 |
| ROOM\_REVIEWS | 5149 |
| ROOM LOCATION | 7907 |
| NEIGHBOURHOOD\_REGION | 43 |

Some of the data cleaning activities performed are as below

1. Host: Duplicate entries removed and host\_name with Nan is updated with “Unknown” (**name.py** and **host.py**)

def cleanName(data):  
 data = data.fillna("Unknown")  
 data = data.replace(""**,** "Unknown")  
 return data

def clean\_data(self**,**df):  
 df = df.drop\_duplicates(subset=[Listing\_constants.host\_id])  
 return df

1. Room\_reviews: Removed entries which didn’t have any reviews (**room\_reviews.py**)

def clean\_data(self**,**data):  
 new\_data = data.dropna().reset\_index(drop=True)  
 return new\_data

1. Neighbourhood\_region: Removed the Duplicate entries

def clean\_data(self**,**data):  
 data = data.drop\_duplicates(subset=[Listing\_constants.neighbourhood])  
 return data

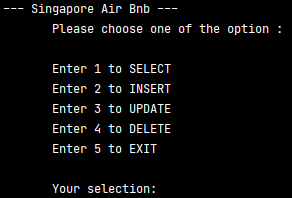
## User operations and executions

* Once the data is normalised and store customer is presented with an option to perform CRUD operations, Currently this functionality is for SELECT / UPDATE / INSERT / DELETE. This can be expanded for other operations like CREATE or DROP or ALTER of tables and databases.

The input is taken from the user and necessary CRUD operations are performed. User has also options exit the execution without performing any operation.

As dictionary is maintain in code and the keys of the dictionary are matched with the entry provided by the user and based on the match respected object gets instantiated.

self.operation\_dic = {'1': Select**,** '2': Insert**,** '3': Update**,** '4': Delete**,** '5': "EXIT"}

****

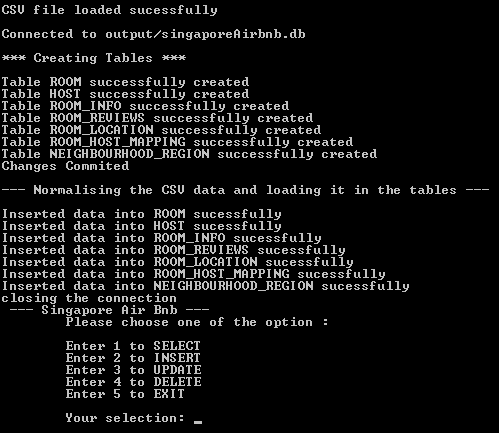
If customer enters anything others than 1,2,3,4 or 5 then the options are again prepopulated to take correct entry from the user.

If the user chooses to INSERT / DELETE / UPDATE then the query is executed, if any error occurs then it is populated on console.

If SELECT is chosen, once the query gets executed successfully then first 5 rows are shown on the console and the complete data is stored in “**Output\_Result.csv**” under output directory

## Quick look at console

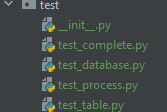




****

## Unit Testing

The **unittest** framework which is already included in python packages is used to perform unit testing here and a separate test directory is created to store all the test cases.



test\_database.py holds all the test case for the code present in database package under src

* Test cases for data base connections

test\_process.py holds all the test case for the code present in process package under src

* Test cases for file operations
* Test cases for query executions
* Test cases for customer inputs for CRUD operations

test\_tables.py holds all the test case for the code present in tables package under src

* Test cases for data generation of each table
* Test cases for data cleaning for each table

test\_complete.py has a test suite built to run all the test cases at once

## Source Code

## Main.py

**Path: src/main.py**

from src.process.listingUtil import ListingUtil  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 '''This is the method used for execution of the code'''  
 listing = ListingUtil()  
 listing.main()

## Package process

## listingConstants.py

**Path: src/process/listingConstants.py**

class Listing\_constants():  
  
 csv\_path = "../csvData/listings.csv"  
 db\_name = "../output/singaporeAirbnb.db"  
 output\_result = "../output/Output\_Result.csv"  
 room\_id="id"  
 host\_id = "host\_id"  
 room\_name='name'  
 host\_name="host\_name"  
 host\_list\_count="calculated\_host\_listings\_count"  
 room\_type = "room\_type"  
 price = "price"  
 min\_nights="minimum\_nights"  
 availability = "availability\_365"  
 neighbourhood\_group="neighbourhood\_group"  
 neighbourhood = "neighbourhood"  
 latitude = "latitude"  
 longitude = "longitude"  
 for\_room="room"  
 no\_reviews = 'number\_of\_reviews'  
 last\_review = "last\_review"  
 review\_per\_month = "reviews\_per\_month"  
 MENU\_PROPMT = """\n\t\t\t\t--- Singapore Air Bnb ---  
 Please choose one of the option :  
   
 Enter 1 to SELECT  
 Enter 2 to INSERT  
 Enter 3 to UPDATE  
 Enter 4 to DELETE  
 Enter 5 to EXIT  
  
 Your selection: """  
 QUERY\_PROMPT = "Please enter your query::\n"  
  
 db\_table\_column\_name={'room' : ['ID'**,** 'NAME']**,** 'host' : ['ID'**,** 'NAME'**,** 'LISTING\_COUNT']**,** 'room\_host\_mapping': ['ROOM\_ID'**,** 'HOST\_ID']**,** 'room\_location' : ['ROOM\_ID'**,**'LATITUDE'**,**'LONGITUDE'**,**'NEIGHBOURHOOD']**,** 'room\_reviews' : ['ROOM\_ID'**,** 'NO\_OF\_REVIEWS'**,** 'LAST\_REVIEW'**,** 'REVIEW\_PER\_MONTH']**,** 'room\_info' : ['ROOM\_ID'**,**'ROOM\_TYPE'**,**'ROOM\_PRICE'**,**'MINIMUM\_NIGHTS'**,**'AVAILABILITY']**,** 'neighbourhood\_region' : ['NEIGHBOURHOOD'**,**'REGION']}

## queryConstant.py

**Path: src/process/queryConstant.py**

class Query\_Constants:  
  
 create\_room= "CREATE TABLE ROOM (ID INTEGER PRIMARY KEY NOT NULL UNIQUE, " \  
 "NAME VARCHAR(30) NOT NULL)"  
  
 create\_host= "CREATE TABLE HOST (ID INTEGER PRIMARY KEY NOT NULL UNIQUE, " \  
 "NAME VARCHAR(30) NOT NULL," \  
 "LISTING\_COUNT INTEGER NOT NULL)"  
  
 create\_room\_reviews= "CREATE TABLE ROOM\_REVIEWS(ROOM\_ID INTEGER NOT NULL, " \  
 "NO\_OF\_REVIEWS INTEGER NOT NULL, " \  
 "LAST\_REVIEW DATE NOT NULL," \  
 "REVIEW\_PER\_MONTH FLOAT NOT NULL, " \  
 "FOREIGN KEY (ROOM\_ID) REFERENCES ROOM(ID))"  
  
 create\_room\_info= "CREATE TABLE ROOM\_INFO(ROOM\_ID INTEGER NOT NULL," \  
 "ROOM\_TYPE VARCHAR(30) NOT NULL, " \  
 "ROOM\_PRICE FLOAT NOT NULL," \  
 "MINIMUM\_NIGHTS INTEGER NOT NULL," \  
 "AVAILABILITY INTEGER NOT NULL," \  
 "FOREIGN KEY (ROOM\_ID) REFERENCES ROOM(ID))"  
  
 create\_room\_location= "CREATE TABLE ROOM\_LOCATION(ROOM\_ID INTEGER NOT NULL, " \  
 "LATITUDE Float NOT NULL, " \  
 "LONGITUDE VARCHAR(30) NOT NULL, " \  
 "NEIGHBOURHOOD VARCHAR(30) NOT NULL, " \  
 "FOREIGN KEY (NEIGHBOURHOOD) REFERENCES NEIGHBOURHOOD\_REGION(NEIGHBOURHOOD)," \  
 "FOREIGN KEY (ROOM\_ID) REFERENCES ROOM(ID))"  
  
  
 create\_room\_host\_mapping= "CREATE TABLE ROOM\_HOST\_MAPPING(ROOM\_ID INTEGER NOT NULL UNIQUE," \  
 "HOST\_ID INTEGER NOT NULL," \  
 "FOREIGN KEY (ROOM\_ID) REFERENCES ROOM(ID)," \  
 "FOREIGN KEY (HOST\_ID) REFERENCES HOST(ID))"  
  
  
 create\_neighbourhood\_region = "CREATE TABLE NEIGHBOURHOOD\_REGION(NEIGHBOURHOOD VARCHAR(30) PRIMARY KEY NOT NULL UNIQUE, " \  
 "REGION VARCHAR(30) NOT NULL)"

## listingUtil.py

**Path: src/process/listingUtil.py**

import pandas as pd  
import os  
import sys  
from src.database.database\_operations import Database  
from src.database.insert import Insert  
from src.process.listingConstants import Listing\_constants  
from src.database.create import Create  
from src.database.update import Update  
from src.database.delete import Delete  
from src.database.select import Select  
  
  
class ListingUtil:  
  
 def \_\_init\_\_(self):  
 *""" This class supports all the operation required for the normalisation and retrieval of data"""* self.operation = None  
 self.query = None  
 self.execute = True  
 self.operation\_dic = {'1': Select**,** '2': Insert**,** '3': Update**,** '4': Delete**,** '5': "EXIT"}  
 pass  
  
 def main(self):  
 self.cleanFiles(Listing\_constants.db\_name)  
 self.create\_load\_database(self.load\_csv()**,** Database(None).connect(Listing\_constants.db\_name))  
 while(self.execute):  
 if (self.getJobInputs()):  
 self.perform\_operation(self.operation**,** self.query)  
  
 @staticmethod  
 def cleanFiles(file):  
 *"""Removes any db file if already exist"""* if os.path.exists(file):  
 os.remove(file)  
  
 def create\_load\_database(self**,**listing\_df**,** conn):  
 *"""Creates the normalised tables and inserts the data as per the csv given"""* create = Create(conn)  
 create.create\_Tables()  
 create.commit()  
 insert = Insert(conn)  
 insert.load\_Tables(listing\_df)  
 insert.commit()  
 create.closeConnection()  
  
 def perform\_operation(self**,** operation**,** query):  
 *"""Executes the query given by user on console"""* conn = Database(None).connect(Listing\_constants.db\_name)  
 task = operation(conn)  
 task.execute\_query(query)  
 task.commit()  
 task.closeConnection()  
  
  
 def getJobInputs(self):  
 *"""User experience to select the operation to be performed """* val = input(Listing\_constants.MENU\_PROPMT)  
 if ('EXIT' == self.operation\_dic.get(val)):  
 return sys.exit(**1**)  
 elif (None != self.operation\_dic.get(val)):  
 self.query = input(Listing\_constants.QUERY\_PROMPT)  
 self.operation = self.operation\_dic.get(val)  
 return self.validateQuery(self.query**,** self.operation)  
 else:  
 print("\nInvalid value entered\n")  
 return False  
  
 @staticmethod  
 def validateQuery(query**,** oper):  
 if (query.lower().find(oper.\_\_name\_\_.lower()) == **0**):  
 return True  
 else:  
 print(f"Error ::::: Query entered is not {oper.\_\_name\_\_}")  
 return False  
  
  
 @staticmethod  
 def load\_csv():  
 file = pd.read\_csv(Listing\_constants.csv\_path)  
 print('\n\nCSV file loaded sucessfully\n')  
 return file

## Package "database”

## database\_operations.py

**Path: src/database/database\_operations.py**

import sqlite3  
import sys  
  
class Database:  
  
 def \_\_init\_\_(self**,**conn):  
 self.conn = conn  
 self.df\_file = None  
  
 def connect(self**,**df\_file):  
 *""" create a database connection to a SQLite database """* self.df\_file=df\_file  
 try:  
 self.conn = sqlite3.connect(self.df\_file)  
 print(f"Connected to {self.df\_file} \n")  
 return self.conn  
 except Exception as ex:  
 print (ex)  
 return sys.exit()  
  
 def closeConnection(self):  
 *""" Closing a database connection in SQLite database """* print ("Connection Closed")  
 if self.conn:  
 self.conn.close()  
  
 def commit(self):  
 *""" Commiting the changes """* print("Changes Commited")  
 self.conn.commit()  
  
 def rollback(self):  
 *""" Rolling back the changes """* print ("Rolling back all the changes")  
 self.conn.rollback()

## crud\_operations.py

**Path: src/database/crud\_operations.py**

from sqlite3 import Error  
from src.database.database\_operations import Database  
  
  
class Crud(Database):  
  
 def \_\_init\_\_(self**,**conn):  
 super().\_\_init\_\_(conn)  
 self.conn = conn  
 self.query = None  
 self.query\_results = None  
  
 def execute\_query(self**,**query):  
 *"""Execute the query for all the SQL queries given in console"""* self.query=query  
 try:  
 c= self.conn.cursor()  
 self.query\_results = c.execute(self.query)  
 except Error as e:  
 print (f"Error in Query::::{e}")  
 except TypeError as e:  
 print(e)  
  
  
  
 def execute\_dataframe\_query(self**,** df**,** table**,** conn):  
 *"""Direct insertion of Dataframe to SQL"""* df.to\_sql(table.\_\_name\_\_.upper()**,** con=conn**,** if\_exists='append'**,** index=False)  
 print(f"Inserted data into {table.\_\_name\_\_.upper()} sucessfully")

## create.py

**Path: src/database/create.py**

from src.tables.room import Room  
from src.tables.host import Host  
from src.tables.room\_info import Room\_info  
from src.tables.room\_reviews import Room\_reviews  
from src.database.crud\_operations import Crud  
from src.tables.room\_location import Room\_location  
from src.tables.room\_host\_mapping import Room\_host\_mapping  
from src.tables.neighbourhood\_region import Neighbourhood\_region  
  
class Create(Crud):  
  
 def \_\_init\_\_(self**,**conn):  
 super().\_\_init\_\_(conn)  
 self.conn=conn  
 self.tables = [Room**,** Host**,** Room\_info**,** Room\_reviews**,** Room\_location**,** Room\_host\_mapping**,**Neighbourhood\_region]  
 print("\*\*\* Creating Tables \*\*\*\n")  
  
 def create\_Tables(self):  
 self.run\_table\_query(self.tables**,**self.conn)  
  
 def run\_table\_query(self**,** tableList**,** conn):  
 for table in tableList:  
 Crud(conn).execute\_query(table.query())  
 print(f"Table {table.\_\_name\_\_.upper()} successfully created")

## insert.py

**Path: src/database/insert.py**

from src.tables.room import Room  
from src.tables.host import Host  
from src.tables.room\_info import Room\_info  
from src.tables.room\_reviews import Room\_reviews  
from src.database.crud\_operations import Crud  
from src.tables.room\_location import Room\_location  
from src.tables.room\_host\_mapping import Room\_host\_mapping  
from src.tables.neighbourhood\_region import Neighbourhood\_region  
  
  
class Insert(Crud):  
  
 def \_\_init\_\_(self**,**conn):  
 super().\_\_init\_\_(conn)  
 self.conn=conn  
 self.tables =[Room**,** Host**,** Room\_info**,** Room\_reviews**,** Room\_location**,** Room\_host\_mapping**,**Neighbourhood\_region]  
  
  
 def load\_Tables(self**,**data):  
 print("\n--- Normalising the CSV data and loading it in the tables ---\n")  
 self.load\_data(self.tables**,** data)  
  
 def load\_data (self**,** tableList**,** data):  
 for table in tableList:  
 df = self.generateData(table(data))  
 df.columns = table(data).getColNames()  
 super().execute\_dataframe\_query(df**,**table**,**self.conn)  
  
 def generateData(self**,** table):  
 return table.clean\_data(table.getData())

## select.py

**Path: src/database/select.py**

from src.database.crud\_operations import Crud  
from src.process.listingConstants import Listing\_constants  
import pandas as pd  
  
  
class Select(Crud):  
  
 def \_\_init\_\_(self**,**conn):  
 super().\_\_init\_\_(conn)  
 self.conn=conn  
  
 def execute\_query(self**,**query):  
 try:  
 result\_df = pd.read\_sql(query**,** con=self.conn)  
 print (f"First five entries printed...\n{result\_df.head(**5**)}")  
 result\_df.to\_csv(Listing\_constants.output\_result)  
 print (f'\nResult copied to "../output/Output.Result.csv" in project folder')  
 except Exception as ex:  
 print(ex)

## update.py

**Path: src/database/update.py**

from src.database.crud\_operations import Crud  
  
class Update(Crud):  
 def \_\_init\_\_(self**,** conn):  
 super().\_\_init\_\_(conn)  
 self.conn=conn  
 self.query=None

## delete.py

**Path: src/database/delete.py**

1. from src.database.crud\_operations import Crud  
     
   class Delete(Crud):  
    def \_\_init\_\_(self**,** conn):  
    super().\_\_init\_\_(conn)  
    self.conn=conn  
    self.query=None

## Package "tables”

## id.py

**Path: src/tables/id.py**

from src.process.listingConstants import Listing\_constants  
  
class Id:  
 def \_\_init\_\_(self**,** data**,** type):  
 self.type = type  
 self.data = data  
  
 def id\_type(self):  
 return Listing\_constants.int  
  
  
 def id\_data(self):  
 if self.type == Listing\_constants.for\_room:  
 return self.data.loc[:**,**[Listing\_constants.room\_id]]  
 else:  
 return self.data.loc[:**,**[Listing\_constants.host\_id]]

## name.py

**Path: src/tables/name.py**

from src.process.listingConstants import Listing\_constants  
from src.tables.id import Id  
  
  
class Name(Id):  
 def \_\_init\_\_(self**,** data**,** type):  
 super().\_\_init\_\_(data**,** type)  
 self.data=data  
 self.type=type  
  
 def name\_type(self):  
 return Listing\_constants.text  
  
 def name\_data(self):  
 if self.type == Listing\_constants.for\_room:  
 return self.data.loc[:**,**[Listing\_constants.room\_name]]  
 else:  
 return self.data.loc[:**,**[Listing\_constants.host\_name]]  
  
 @staticmethod  
 def cleanName(data):  
 data = data.fillna("Unknown").replace(""**,** "Unknown")  
 return data

## room.py

**Path: src/tables/room.py**

from src.tables.name import Name  
from src.process.queryConstants import Query\_Constants  
from src.process.listingConstants import Listing\_constants  
import pandas as pd  
  
class Room(Name):  
 def \_\_init\_\_(self**,** data):  
 super().\_\_init\_\_(data**,** Listing\_constants.for\_room)  
 self.data = data  
  
 def getData(self):  
 return pd.concat([self.id\_data()**,**self.name\_data()]**,** axis=**1**)  
  
 def clean\_data(self**,** df):  
 df[Listing\_constants.room\_name] = self.cleanName(df[Listing\_constants.room\_name])  
 return df  
  
 def getColNames(self):  
 return Listing\_constants.db\_table\_column\_name.get("room")  
  
 @staticmethod  
 def query():  
 return Query\_Constants.create\_room

## host.py

**Path: src/tables/host.py**

from src.tables.name import Name  
from src.process.queryConstants import Query\_Constants  
from src.process.listingConstants import Listing\_constants  
import pandas as pd  
  
  
class Host(Name):  
 def \_\_init\_\_(self**,** data):  
 super().\_\_init\_\_(data**,** "host")  
 self.data = data  
  
 def getData(self):  
 return pd.concat([self.id\_data()**,** self.name\_data()**,** self.hostListCount\_Data()]**,** axis=**1**)  
  
 def hostListCount\_Data(self):  
 return self.data.loc[:**,** [Listing\_constants.host\_list\_count]]  
  
 def clean\_data(self**,** df):  
 df = df.drop\_duplicates(subset=[Listing\_constants.host\_id])  
 return df  
  
 def getColNames(self):  
 return Listing\_constants.db\_table\_column\_name.get("host")  
  
 @staticmethod  
 def query():  
 return Query\_Constants.create\_host

## room\_info.py

**Path: src/tables/room\_info.py**

from src.process.listingConstants import Listing\_constants  
from src.process.queryConstants import Query\_Constants  
from src.tables.id import Id  
import pandas as pd  
  
  
class Room\_info(Id):  
 def \_\_init\_\_(self**,** data):  
 super().\_\_init\_\_(data**,** "room")  
 self.data = data  
  
  
 def getData(self):  
 return pd.concat([self.id\_data()**,**self.room\_info\_data()]**,** axis=**1**)  
  
 def room\_info\_data(self):  
 return self.data.loc[:**,** [Listing\_constants.room\_type**,** Listing\_constants.price**,** Listing\_constants.min\_nights**,** Listing\_constants.availability]]  
  
 def getColNames(self):  
 return Listing\_constants.db\_table\_column\_name.get("room\_info")  
  
 def clean\_data(self**,**data):  
 return data  
  
 @staticmethod  
 def query():  
 return Query\_Constants.create\_room\_info

## room\_reviews.py

**Path: src/tables/room\_reviews.py**

from src.tables.id import Id  
from src.process.listingConstants import Listing\_constants  
from src.process.queryConstants import Query\_Constants  
import pandas as pd  
  
  
class Room\_reviews(Id):  
 def \_\_init\_\_(self**,** data):  
 super().\_\_init\_\_(data**,**Listing\_constants.for\_room)  
 self.data=data  
  
 def getData(self):  
 return pd.concat([self.id\_data()**,**self.review\_data()]**,** axis=**1**)  
  
 def review\_data(self):  
 return self.data.loc[:**,**[Listing\_constants.no\_reviews**,** Listing\_constants.last\_review**,** Listing\_constants.review\_per\_month]]  
  
  
 def clean\_data(self**,**data):  
 new\_data = data.dropna().reset\_index(drop=True)  
 return new\_data  
  
 def getColNames(self):  
 return Listing\_constants.db\_table\_column\_name.get("room\_reviews")  
  
 @staticmethod  
 def query():  
 return Query\_Constants.create\_room\_reviews

## room\_location.py

**Path: src/tables/room\_location.py**

from src.tables.id import Id  
from src.process.listingConstants import Listing\_constants  
from src.process.queryConstants import Query\_Constants  
import pandas as pd  
  
class Room\_location(Id):  
 def \_\_init\_\_ (self**,**data):  
 super().\_\_init\_\_(data**,** Listing\_constants.for\_room)  
 self.data = data  
  
 def getData(self):  
 return pd.concat([self.id\_data()**,**self.room\_location\_data()]**,** axis=**1**)  
  
 def room\_location\_data(self):  
 return self.data.loc[:**,**[Listing\_constants.latitude**,** Listing\_constants.longitude**,** Listing\_constants.neighbourhood]]  
  
 def getColNames(self):  
 return Listing\_constants.db\_table\_column\_name.get("room\_location")  
 def clean\_data(self**,**data):  
 return data  
  
 @staticmethod  
 def query():  
 return Query\_Constants.create\_room\_location

## room\_host\_mapping.py

**Path: src/tables/room\_host\_mapping.py**

from src.tables.id import Id  
from src.process.listingConstants import Listing\_constants  
from src.process.queryConstants import Query\_Constants  
import pandas as pd  
  
class Room\_host\_mapping(Id):  
 def \_\_init\_\_ (self**,**data):  
 super().\_\_init\_\_(data**,** Listing\_constants.for\_room)  
 self.data = data  
 self.col\_names = ['ROOM\_ID'**,** 'HOST\_ID']  
  
  
 def getData(self):  
 return pd.concat([self.id\_data()**,**self.data[Listing\_constants.host\_id]]**,** axis=**1**)  
  
 def getColNames(self):  
 return Listing\_constants.db\_table\_column\_name.get("room\_host\_mapping")  
  
 def clean\_data(self**,**data):  
 return data  
  
 @staticmethod  
 def query():  
 return Query\_Constants.create\_room\_host\_mapping

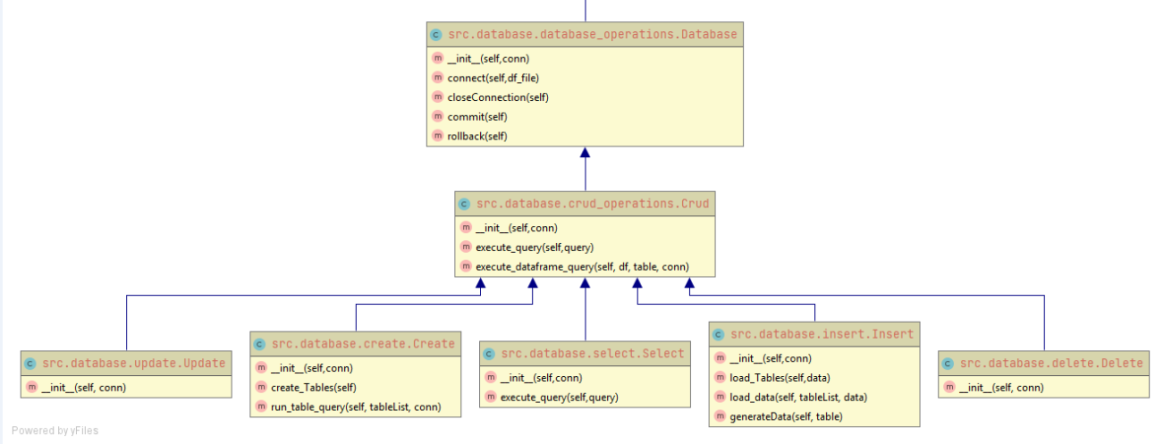
## neighbourhood\_region.py

**Path: src/tables/neighbourhood\_region.py**

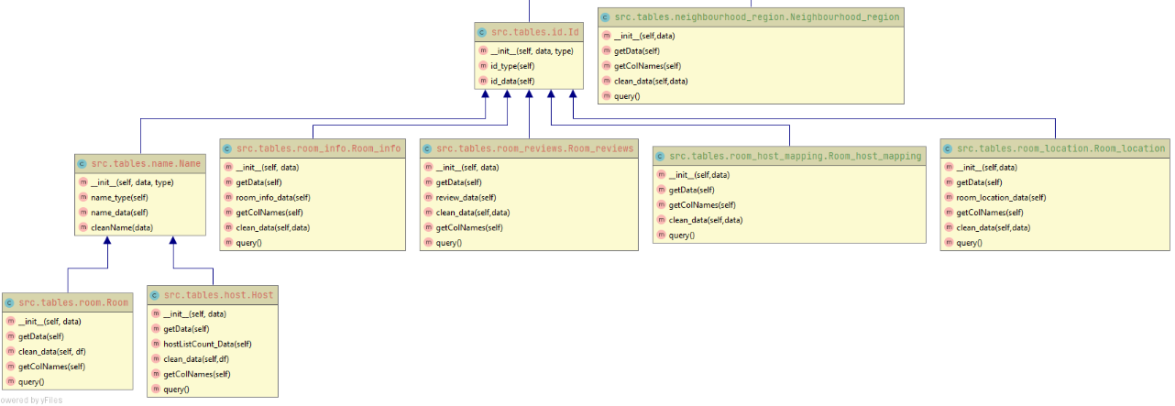
from src.process.listingConstants import Listing\_constants  
from src.process.queryConstants import Query\_Constants  
import pandas as pd  
  
class Neighbourhood\_region():  
 def \_\_init\_\_ (self**,**data):  
 self.data = data  
  
 def getData(self):  
 return self.data.loc[:**,**[Listing\_constants.neighbourhood**,**Listing\_constants.neighbourhood\_group]]  
  
 def getColNames(self):  
 return Listing\_constants.db\_table\_column\_name.get("neighbourhood\_region")  
  
 def clean\_data(self**,**data):  
 data = data.drop\_duplicates(subset=[Listing\_constants.neighbourhood])  
 return data  
  
 @staticmethod  
 def query():  
 return Query\_Constants.create\_neighbourhood\_region

## UML Diagram

**Package database**

****

**Package tables**

****

**Package process**



## Test Code

## test\_process.py

from src.process.listingUtil import ListingUtil  
import os.path  
import tempfile  
import unittest  
from unittest.mock import patch  
  
  
class TestProcess(unittest.TestCase):  
 tmpfilepath = os.path.join(tempfile.gettempdir()**,** "tmp-testfile")  
 def setUp(self):  
 with open(self.tmpfilepath**,**'w') as f:  
 f.write("Now the file has more content!")  
  
 def test\_cleanFiles(self):  
 ListingUtil.cleanFiles(self.tmpfilepath)  
 self.assertFalse(os.path.isfile(self.tmpfilepath)**,** "Failed to remove the file.")  
  
 @patch('builtins.input'**,** side\_effect=['1'**,** 'Select \* from HOST'])  
 def test\_getJobInputs1(self**,** input):  
 self.assertEqual(ListingUtil().getJobInputs()**,** True)  
  
  
 @patch('builtins.input'**,** side\_effect=['2'**,** 'Insert into host("ID","NAME") values (1,"newEntry")'])  
 def test\_getJobInputs2(self**,** input):  
 self.assertEqual(ListingUtil().getJobInputs()**,** True)  
  
  
 @patch('builtins.input'**,** side\_effect=['4'**,** 'Delete from host where id = 1'])  
 def test\_getJobInputs3(self**,** input):  
 self.assertEqual(ListingUtil().getJobInputs()**,** True)  
  
 @patch('builtins.input'**,** side\_effect=['5'])  
 def test\_getJobInputs4(self**,** input):  
 with self.assertRaises(SystemExit) as cm:  
 ListingUtil().getJobInputs()  
 self.assertEqual(cm.exception.code**, 1**)  
  
 @patch('builtins.input'**,** side\_effect=['10'])  
 def test\_getJobInputs5(self**,** input):  
 self.assertEqual(ListingUtil().getJobInputs()**,** False)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

## test\_table.py

import unittest  
import pandas as pd  
from src.tables.host import Host  
from src.tables.room import Room  
from src.tables.room\_location import Room\_location  
from src.tables.room\_info import Room\_info  
from src.tables.room\_reviews import Room\_reviews  
from src.tables.room\_host\_mapping import Room\_host\_mapping  
from src.tables.neighbourhood\_region import Neighbourhood\_region  
from pandas.testing import assert\_frame\_equal  
  
  
class TestTable(unittest.TestCase):  
 def setUp(self):  
 self.dict = {'id': {**0**: **49091, 1**: **50646, 2**: **56334**}**,** 'name': {**0**: 'COZICOMFORT LONG TERM STAY ROOM 2'**,1**: 'Pleasant Room along Bukit Timah'**,2**: 'COZICOMFORT'}**,** 'host\_id': {**0**: **266763, 1**: **227796, 2**: **266763**}**,** 'host\_name': {**0**: 'Francesca'**, 1**: 'Sujatha'**, 2**: 'Francesca'}**,** 'neighbourhood\_group': {**0**: 'North Region'**,1**: 'Central Region'**,2**: 'North Region'}**,** 'neighbourhood': {**0**: 'Woodlands'**, 1**: 'Bukit Timah'**, 2**: 'Woodlands'}**,** 'latitude': {**0**: **1.44255, 1**: **1.33235, 2**: **1.44246**}**,** 'longitude': {**0**: **103.7958, 1**: **103.78520999999999, 2**: **103.79666999999999**}**,** 'room\_type': {**0**: 'Private room'**, 1**: 'Private room'**, 2**: 'Private room'}**,** 'price': {**0**: **83, 1**: **81, 2**: **69**}**,** 'minimum\_nights': {**0**: **180, 1**: **90, 2**: **6**}**,** 'number\_of\_reviews': {**0**: **1, 1**: **18, 2**: **20**}**,** 'last\_review': {**0**: '10/21/2013'**, 1**: '12/26/2014'**, 2**: '10/1/2015'}**,** 'reviews\_per\_month': {**0**: **0.01, 1**: **0.28, 2**: **0.2**}**,** 'calculated\_host\_listings\_count': {**0**: **2, 1**: **1, 2**: **2**}**,** 'availability\_365': {**0**: **365, 1**: **365, 2**: **365**}}  
 self.data = pd.DataFrame(self.dict)  
  
  
 def test\_room\_data(self):  
 obj =Room(self.data)  
 assert\_frame\_equal(obj.getData()**,** self.data[['id'**,**'name']])  
  
 def test\_host\_data(self):  
 obj =Host(self.data)  
 assert\_frame\_equal(obj.getData()**,** self.data[['host\_id'**,**'host\_name'**,**'calculated\_host\_listings\_count']])  
  
 def test\_room\_info\_data(self):  
 obj = Room\_info(self.data)  
 assert\_frame\_equal(obj.getData()**,**self.data[['id'**,**'room\_type'**,** 'price'**,**'minimum\_nights'**,** 'availability\_365']])  
  
 def test\_room\_review\_data(self):  
 obj = Room\_reviews(self.data)  
 assert\_frame\_equal(obj.getData()**,** self.data[['id'**,**'number\_of\_reviews'**,** 'last\_review'**,** 'reviews\_per\_month']])  
  
 def test\_room\_host\_mapping(self):  
 obj = Room\_host\_mapping(self.data)  
 assert\_frame\_equal(obj.getData()**,** self.data[['id'**,**'host\_id']])  
  
  
 def test\_room\_location(self):  
 obj = Room\_location(self.data)  
 assert\_frame\_equal(obj.getData()**,** self.data[['id' **,** 'latitude'**,**'longitude'**,** 'neighbourhood']])  
  
 def test\_neighbourhood\_region(self):  
 obj = Neighbourhood\_region(self.data)  
 assert\_frame\_equal(obj.getData()**,** self.data[['neighbourhood'**,**'neighbourhood\_group']])  
 #  
  
 def test\_clean\_data\_room(self):  
 dict1= {'host\_id' : [**266763,1017645,367042,266763**]**,** 'host\_name' : ['Francesca'**,** 'Bianca' **,** 'Belinda' **,** 'Francesca']}  
 dict2= {'host\_id' : [**266763,1017645,367042**]**,** 'host\_name' : ['Francesca'**,** 'Bianca' **,** 'Belinda']}  
 actual\_data = pd.DataFrame(dict1)  
 expected\_data = pd.DataFrame(dict2)  
 obj = Host(self.data)  
 assert\_frame\_equal(obj.clean\_data(actual\_data)**,** expected\_data)  
  
  
 def test\_clean\_data\_host(self):  
 dict1= {'id' : [**49091,50646,56334,71903**]**,** 'name' : ['Room1'**,** 'Room2' **,** None **,** ""]}  
 dict2 = {'id': [**49091, 50646, 56334, 71903**]**,** 'name': ['Room1'**,** 'Room2'**,** 'Unknown'**,** 'Unknown']}  
 actual\_data = pd.DataFrame(dict1)  
 expected\_data = pd.DataFrame(dict2)  
 obj = Room(self.data)  
 assert\_frame\_equal(obj.clean\_data(actual\_data)**,** expected\_data)  
  
 def test\_clean\_neighbourhood\_region1(self):  
 dict1 = {'neighbourhood\_group' : ['East Region'**,** 'North Region' **,** 'North Region' **,** 'North Region']**,** 'neighbourhood':['Tampines'**,**'Newton'**,**'Newton'**,**'Newton']}  
 dict2 = {'neighbourhood\_group': ['East Region'**,** 'North Region']**,** 'neighbourhood': ['Tampines'**,**'Newton']}  
 actual\_data = pd.DataFrame(dict1)  
 expected\_data = pd.DataFrame(dict2)  
 obj = Neighbourhood\_region(self.data)  
 assert\_frame\_equal(obj.clean\_data(actual\_data)**,** expected\_data)  
  
  
 def test\_clean\_neighbourhood\_region2(self):  
 dict1 = {'neighbourhood\_group' : ['East Region'**,** 'North Region' **,** 'North Region' **,** 'North Region']**,** 'neighbourhood':['Tampines'**,**'Newton'**,**'Bedok'**,**'Ang Mo Kio']}  
 dict2 = {'neighbourhood\_group': ['East Region'**,** 'North Region'**,** 'North Region'**,** 'North Region']**,** 'neighbourhood': ['Tampines'**,** 'Newton'**,** 'Bedok'**,** 'Ang Mo Kio']}  
 actual\_data = pd.DataFrame(dict1)  
 expected\_data = pd.DataFrame(dict2)  
 obj = Neighbourhood\_region(self.data)  
 assert\_frame\_equal(obj.clean\_data(actual\_data)**,** expected\_data)  
  
  
  
 def test\_clean\_data\_room\_reviews(self):  
 dict1 = {'id': [**49091, 50646, 56334, 241510**]**,** 'number\_of\_reviews' : [**10,0,0,3**]**,** 'last\_review':['10/21/2013'**,** None**,** None**,** '7/28/2019']**,** 'reviews\_per\_month':[**0.10,**None**,** None**,0.02**]}  
 dict2 = {'id': [**49091, 241510**]**,** 'number\_of\_reviews': [**10, 3**]**,** 'last\_review': ['10/21/2013'**,**'7/28/2019']**,** 'reviews\_per\_month': [**0.10, 0.02**]}  
 actual\_data = pd.DataFrame(dict1)  
 expected\_data = pd.DataFrame(dict2)  
 obj = Room\_reviews(self.data)  
 assert\_frame\_equal(obj.clean\_data(actual\_data)**,** expected\_data)  
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

## test\_database.py

from src.database.database\_operations import Database  
from src.database.crud\_operations import Crud  
import sqlite3  
from unittest.mock import MagicMock**,**Mock  
import unittest  
import io  
import sys  
  
class TestDatabase(unittest.TestCase):  
  
 def setUp(self):  
 suppress\_text = io.StringIO()  
 sys.stdout = suppress\_text  
  
  
 def test\_sqlite3\_connect\_success(self):  
 sqlite3.connect = MagicMock(return\_value='connection succeeded')  
 conn = Database(None).connect("test\_database")  
 sqlite3.connect.assert\_called\_with('test\_database')  
 self.assertEqual(conn**,** 'connection succeeded')  
  
 def test\_sqlite3\_connect\_failure(self):  
 sqlite3.connect = MagicMock(return\_value='connection failure')  
 conn = Database(None).connect("test\_database")  
 sqlite3.connect.assert\_called\_with('test\_database')  
 self.assertEqual(conn**,** 'connection failure')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## test\_complete.py

import unittest  
import io  
import sys  
from test\_process import TestProcess  
from test\_table import TestTable  
from test\_database import TestDatabase  
  
  
def execute():  
 suppress\_text = io.StringIO()  
 sys.stdout = suppress\_text  
 tc1 = unittest.TestLoader().loadTestsFromTestCase(TestProcess)  
 tc2 = unittest.TestLoader().loadTestsFromTestCase(TestTable)  
 tc3 = unittest.TestLoader().loadTestsFromTestCase(TestDatabase)  
 sanity = unittest.TestSuite([tc1**,**tc2**,**tc3])  
 unittest.TextTestRunner().run(sanity)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 execute()

## Executions

* The code can be executed either in Pycharm or console

**Functional code:** src/main.py

**Unit Testing:** test/test\_complete.py

* Some of the queries execution after successful creation and loading of data base are:

|  |
| --- |
| **--- SELECT QUERIES ---**  SELECT \* FROM ROOM  SELECT \* FROM HOST  SELECT \* FROM HOST WHERE ID BETWEEN 23666 AND 165209  SELECT MAX(PRICE), MIN(PRICE) FROM ROOM\_INFO  SELECT \* FROM ROOM INNER JOIN ROOM\_INFO ON ROOM.ID=ROOM\_INFO.ROOM\_ID;  SELECT \* FROM ROOM LEFT JOIN ROOM\_REVIEWS ON ROOM.ID=ROOM\_REVIEWS.ROOM\_ID;  SELECT \* FROM ROOM\_REVIEWS LEFT JOIN ROOM ON ROOM\_REVIEWS.ROOM\_ID=ROOM.ID;  SELECT COUNT(NEIGHBOURHOOD), REGION  FROM HOST\_NEIGHBOURHOOD GROUP BY REGION  SELECT \*  FROM HOST\_NEIGHBOURHOOD  SELECT DISTINCT REGION, NEIGHBOURHOOD  FROM HOST\_NEIGHBOURHOOD  **--- INSERT QUERIES ---**  INSERT INTO ROOM (ID, NAME) VALUES (1000, "Room in Belfast")  **--- DELETE QUERIES ---**  UPDATE ROOM SET NAME = "Room in Belfast Queen" where ID =1000  **--- UPDATE QUERIES ---**  DELETE FROM ROOM WHERE ID=1000 |

## References

<https://docs.python.org/3/library/unittest.html>

<https://www.w3schools.com/sql/>

<https://docs.python.org/3/tutorial/>

<https://pandas.pydata.org/>