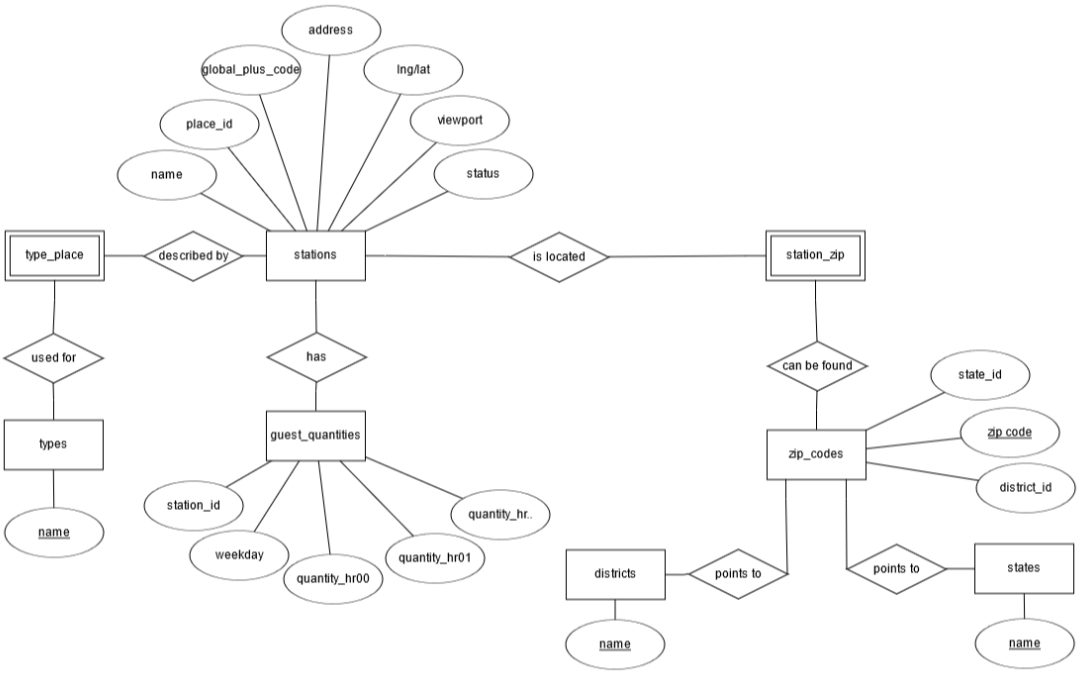
1 Part 1 . . . Database

Since a lot of data can be calculated and the queries for the evaluations should be as fast as possible, the data to be collected should be recorded in a database. A Postgresql DBMS is used. Here I show the basic structure of the database and a little further down my thoughts on the memory requirements and the data types.

1.1 Build up the tables

1. Location types . . . n=100 (`types`) (see: https://developers.google.com/maps/documentation/places/web-service/supported_types)
2. types-stations . . . n= unknown up to 15.000 times x (n:m table)
3. Gas stations . . . n=15,000 (`stations`)
4. station_zip . . . n= max number of places/ stations (n:m table)
5. Peak time profiles . . . n=105.000(`guest_quantity`)
6. Zip codes . . . n=8200 (`zip_codes`)
7. State names . . . n=16(`states`)
8. District names . . . n=401(`districts`)

Below is the entity relationship diagram I tried to implement with the data tables.



1.1.1 Brief consideration about the data types:

There are approx. 15,000 petrol stations (stations) in Germany. For these (at least for most) petrol stations, visitor profiles exist for 24 hours and 7 weekdays. So there are 168 entries per petrol station. Taken together, these 2,520,00 data points represent the largest number of the total storage locations. In order to keep the storage requirements low, these data points are stored as SmallInt, whose individual space requirement is only 2 bytes. Added together, this should results in a requirement of approx. 6 MB for the storage of the visitor profiles. The stations table are cover also a lot of data; this adds up to about 4 MB. One reason ist, the storage needs for google's place_ids cannot be estimated well, as they have no fixed length (and can be very long).

Added to this are the data points of the remaining attributes (place type, postcodes ...). All in all (depending on the data types used and the number of entries), a database space requirement of about 12 MB is expected.

1.1.2 Creating the database

A new database and a user responsible for the project are created on the Postgresql DBMS. The setup is done in the terminal via psql

1. Create a new user
2. ... a new database
3. ... a new schema
4. Grant privileges database and schema to the user

```
superuser=# CREATE USER tankstelle with encrypted password 'verystrongpassword';
CREATE ROLE
superuser=# CREATE DATABASE gas_stations;
CREATE DATABASE
superuser=# \c gas_stations
You are now connected to database "gas_stations" as user "superuser".
gas_stations=# CREATE SCHEMA portfolio;
CREATE SCHEMA
gas_stations=# GRANT ALL PRIVILEGES ON DATABASE gas_stations TO tankstelle;
GRANT
gas_stations=# GRANT ALL PRIVILEGES ON SCHEMA portfolio TO tankstelle;
GRANT
```

I saved the credentials in a Python file `credentials.py` in order to inherit the variables later in the database classes.

1.2 Database class

By calling a GasStationDB instance, the connection to the database is established. The psycopg2 connector and cursor are passed to an instance variable. This means that they can also be called later from other classes. The access data to the database are taken from a file created for this purpose. The class contains functions for creating the above-mentioned tables as well as for deleting tables and deleting rows.

1.2.1 Connect to the db

1.3 Build up the db tables

This code is only needed at the beginning to create the tables of the database.

```
GDB = GasStationDB()

GDB.create_zip_codes()
GDB.create_districs()
GDB.create_states()

GDB.create_types()
GDB.create_stations()
GDB.create_guest_quantities()

GDB.create_types_stations()
GDB.create_station_zip()

#GDB._delete_table('portfolio.###')# if something goes wrong during setup
```

1.4 Doing some initial populatings

Some values, including district names, which are used for the search are to be pre-loaded into the database.

1.4.1 Preparing zip codes, district and state name for populating db

I use an Excel spreadsheet with a listing of public postal codes for Germany. In this table, the respective federal states and names of the districts are also assigned to the postal codes. I will now prepare this table so that I can fill three database tables `zip_codes`, `states` and `districts` with it.

The `types` table is populated during the data fetch for the gas stations when a new type is used as well as the n:m tables `station_zip` and `station_type`.

Getting the lines from the csv file

... and making a list containing zip code, state name, district name and district type of it

```
['PLZ', 'Bundesland', 'Kreis', 'Typ']
['1067', 'Sachsen', 'Dresden', 'Stadt']
```

Processing the feed list

... to a list with full district names (in order to use them for searching via google api). Postal codes in Germany have five digits and some have a zero as the first digit. This zero must be replaced if it has been truncated; the ZIP code is then converted to string. Furthermore two columns are added to the list, one with IDs for the district and one for the state.

Below are some element printed to verify:

```
[['01067', 1, 'Sachsen', 1, 'Stadt Dresden'], ['17237', 6, 'Mecklenburg-
Vorpommern', 50, 'Landkreis Mecklenburgische Seenplatte'], ['28213', 10,
```

```
'Bremen', 98, 'Stadt Bremen'], ['41061', 11, 'Nordrhein-Westfalen', 150, 'Stadt
Mönchengladbach'], ['56594', 13, 'Rheinland-Pfalz', 186, 'Landkreis
Altenkirchen'], ['71576', 16, 'Baden-Württemberg', 258, 'Landkreis Rems-Murr-
Kreis'], ['82299', 14, 'Bayern', 296, 'Landkreis Fürstenfeldbruck'], ['89604',
16, 'Baden-Württemberg', 271, 'Landkreis Alb-Donau-Kreis'], ['97724', 14,
'Bayern', 390, 'Landkreis Rhön-Grabfeld']]
```

Splitting the feed list

... to create single lists for populating the db tables `zip_codes`, `states` and `districts`.

```
[[1, 'Sachsen'], [2, 'Brandenburg'], [3, 'Thüringen'], [4, 'Sachsen-Anhalt'],
[5, 'Berlin'], [6, 'Mecklenburg-Vorpommern'], [7, 'Hamburg'], [8,
'Niedersachsen'], [9, 'Schleswig-Holstein'], [10, 'Bremen'], [11, 'Nordrhein-
Westfalen'], [12, 'Hessen'], [13, 'Rheinland-Pfalz'], [14, 'Bayern'], [15,
'Saarland'], [16, 'Baden-Württemberg']]

[[1, 'Stadt Dresden'], [51, 'Landkreis Vorpommern-Greifswald'], [101, 'Landkreis
Uelzen'], [151, 'Landkreis Viersen'], [201, 'Landkreis Alzey-Worms'], [251,
'Stadt Kaiserslautern'], [301, 'Landkreis Garmisch-Partenkirchen'], [351,
'Landkreis Forchheim'], [401, 'Stadt Eisenach']]

[['01067', 1, 1], ['17237', 6, 50], ['28213', 10, 98], ['41061', 11, 150],
['56594', 13, 186], ['71576', 16, 258], ['82299', 14, 296], ['89604', 16, 271],
['97724', 14, 390]]
```

populating database with zip codes

warning: normally you should not run the code (run this code only once, while setting up the database)

```
cmd = """INSERT INTO portfolio.zip_codes (zip_code, state_id, district_id) VALUES ( %s, %s, %s )
GDB.cur.executemany(cmd, feed_zip)
GDB.conn.commit()
```

populating database with districts

warning: normally you should not run the code (run this code only once, while setting up the database)

```
cmd = """INSERT INTO portfolio.districts (district_id, dist_name) VALUES ( %s, %s );"""
GDB.cur.executemany(cmd, feed_district)
GDB.conn.commit()
```

populating database with states

warning: you should not run the code (run this code only once, while setting up the database)

```
cmd = """INSERT INTO portfolio.states (state_id, state_name) VALUES ( %s, %s );"""
GDB.cur.executemany(cmd, feed_state)
GDB.conn.commit()
```

1.5 Do a couple of simple queries

in order to check the tables ... of course not until the database has been populated with some initial data. Afterwards, when the data collection has been completed, I will determine the storage space required for the database. The values can be found at the bottom of this part. The control queries for the individual tables are listed below.

Check pre-populated tables:

1.5.1 pre-populated zip_codes table

The database was successfully populated with all 8196 zip codes.

Below are some query results to reference:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8196 entries, 0 to 8195
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   zip_id       8196 non-null   int64
 1   zip_code     8196 non-null   object
 2   state_id     8196 non-null   int64
 3   district_id  8196 non-null   int64
dtypes: int64(3), object(1)
memory usage: 256.2+ KB
None
```

	zip_id	zip_code	state_id	district_id
0	1	01067	1	1
2000	2001	28213	10	98
4000	4001	56594	13	186
6000	6001	82299	14	296
8000	8001	97724	14	390

1.5.2 pre-populated districts table

Also all 402 counties / cities can be found in the database.

See some query results to verify:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 402 entries, 0 to 401
Data columns (total 2 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   district_id  402 non-null    int64
 1   dist_name    402 non-null    object
dtypes: int64(1), object(1)
memory usage: 6.4+ KB
None
```

```
     district_id                        dist_name
0              1                     Stadt Dresden
50            51   Landkreis Vorpommern-Greifswald
100          101                  Landkreis Uelzen
150          151                 Landkreis Viersen
200          201              Landkreis Alzey-Worms
250          251               Stadt Kaiserslautern
300          301  Landkreis Garmisch-Partenkirchen
350          351                Landkreis Forchheim
400          401                     Stadt Eisenach
```

1.5.3 pre-populated states table

The 16 federal states have also been loaded into the database.

Also here are some query results to reference:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16 entries, 0 to 15
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   state_id    16 non-null     int64
 1   state_name  16 non-null     object
dtypes: int64(1), object(1)
memory usage: 384.0+ bytes
None
```

```
    state_id         state_name
0          1            Sachsen
3          4     Sachsen-Anhalt
6          7            Hamburg
9         10             Bremen
12        13    Rheinland-Pfalz
15        16  Baden-Württemberg
```

1.6 Check collected data

After the database has been populated with values from the web search, you can see the values inside the other tables. Some control queries are shown here to check whether the tables and the relationships work.

1.6.1 collected locations/stations

Here I check whether the values from the Google Place API have been inserted into the table completely and in the correct formatting. If not, an error exception is thrown and I have to modify the data processing (more details in the next part). The data types below correspond to the pandas dataframe and not to those from the DB table:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9815 entries, 0 to 9814
Data columns (total 12 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   station_id                       9815 non-null   int64
 1   name                             9815 non-null   object
 2   address                          9815 non-null   object
 3   place_id                         9815 non-null   object
 4   global_plus_code                 9813 non-null   object
 5   location_lat                     9815 non-null   object
 6   location_lng                     9815 non-null   object
 7   location_viewport_northeast_lat  9815 non-null   object
 8   location_viewport_northeast_lng  9815 non-null   object
 9   location_viewport_southwest_lat  9815 non-null   object
 10  location_viewport_southwest_lng  9815 non-null   object
 11  business_status                  9814 non-null   object
dtypes: int64(1), object(11)
memory usage: 920.3+ KB
None
```

```
      station_id                      name  \
0              1  TotalEnergies Tankstelle
4000        4001        Markant Tankstelle
8000        8000              T-Tankstelle

                                           address  \
0             Hamburger Str. 44, 01067 Dresden, Germany
4000           Wittmunder Str. 20, 26441 Jever, Germany
8000  Augsburger Str. 61, 91781 Weißenburg in Bayern...

                        place_id        global_plus_code   location_lat  \
0     ChIJm-JGZ5HPCUcRvht1l3CMwgk  9F3M3M6V+GG             51.0612678000
4000  ChIJHdQvXG-JtkcReGatcPJ9gpA  9F59HVFV+VC             53.5747415000
8000  ChIJ_yiTCrw6n0cRkYA0w0NHQvM  8FXG2XC8+GC             49.0213062000

       location_lng location_viewport_northeast_lat  \
0     13.6938440000               51.06261762989272
4000   7.8935165000               53.57604497989271
8000  10.9660309000               49.02273852989272

     location_viewport_northeast_lng location_viewport_southwest_lat  \
0                  13.69519382989272               51.05991797010728
4000                7.89483552989272               53.57334532010727
8000               10.96721372989272               49.02003887010728

     location_viewport_southwest_lng         business_status
```

```
0                  13.69249417010728  OPERATIONAL
4000                7.89213587010728  OPERATIONAL
8000               10.96451407010728  OPERATIONAL
```

1.6.2 collected relative guest quantities / peak times

The data here I read out for the same purpose as for stations; the troubleshooting procedure is also the same:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67465 entries, 0 to 67464
Data columns (total 27 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   quant_id    67465 non-null  int64
 1   station_id  67465 non-null  int64
 2   weekday     67465 non-null  int64
 3   hr00        50224 non-null  float64
 4   hr01        50224 non-null  float64
... ... ... ... ...
 25  hr22        50224 non-null  float64
 26  hr23        50224 non-null  float64
dtypes: float64(24), int64(3)
memory usage: 13.9 MB
None
```

	quant_id	station_id	weekday	hr00	hr01	hr02	hr03	hr04	hr05	\
0	1	1	7	0.0	0.0	0.0	0.0	0.0	0.0	
10000	9976	1442	7	1.0	1.0	1.0	1.0	1.0	1.0	
20000	19945	2885	3	11.0	11.0	11.0	11.0	14.0	18.0	
30000	29996	4381	3	0.0	0.0	0.0	0.0	0.0	0.0	
40000	40000	5834	6	0.0	0.0	0.0	0.0	0.0	0.0	
50000	50001	7291	1	2.0	2.0	2.0	2.0	2.0	2.0	
60000	59990	8735	2	2.0	2.0	2.0	2.0	2.0	2.0	

	hr06	...	hr14	hr15	hr16	hr17	hr18	hr19	hr20	hr21	hr22	hr23
0	1.0	...	72.0	70.0	63.0	52.0	39.0	26.0	14.0	5.0	0.0	0.0
10000	1.0	...	97.0	100.0	92.0	77.0	57.0	37.0	18.0	4.0	1.0	1.0
20000	22.0	...	44.0	40.0	40.0	37.0	33.0	29.0	25.0	18.0	14.0	11.0
30000	0.0	...	39.0	66.0	100.0	65.0	30.0	21.0	16.0	0.0	0.0	0.0
40000	11.0	...	53.0	47.0	38.0	29.0	22.0	16.0	11.0	7.0	3.0	0.0
50000	12.0	...	34.0	63.0	91.0	95.0	72.0	36.0	4.0	2.0	2.0	2.0
60000	5.0	...	67.0	71.0	70.0	64.0	54.0	41.0	27.0	15.0	5.0	2.0

[7 rows x 27 columns]

1.6.3 resulting station_zip table

This query is used to check the relationships between the `stations` and `zip_codes` tables.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9779 entries, 0 to 9778
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   zds_station_id  9779 non-null   int64
 1   zds_zip_id      9779 non-null   int64
dtypes: int64(2)
memory usage: 152.9 KB
None
```

	zds_station_id	zds_zip_id
0	1	1
2000	2005	3059
4000	4015	1870
6000	6020	7021
8000	8027	6500

1.6.4 collected types

The values for the `types` are read in during data gathering, one by one. Here I check if the values are added correctly.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61 entries, 0 to 60
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   type_id  61 non-null     int64
 1   name     61 non-null     object
dtypes: int64(1), object(1)
memory usage: 1.1+ KB
None
```

	type_id	name
0	1	gas_station
10	11	atm
20	21	laundry
30	31	lodging
40	41	campground
50	51	clothing_store
60	61	travel_agency

1.6.5 resulting station_type table

This is the table to check if the linking between `stations` and `types` table works.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50395 entries, 0 to 50394
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   ts_types_id     50395 non-null  int64
 1   ts_stations_id  50395 non-null  int64
dtypes: int64(2)
memory usage: 787.5 KB
None
```

	ts_types_id	ts_stations_id
0	1	1
10000	5	1854
20000	6	3843
30000	6	6085
40000	9	8314
50000	8	9690

1.6.6 database memory requirements

Here are the values for the memory requirements of the individual tables. In total, the originally calculated values already come together for about 2/3 of the expected data sets.

```
types: 24 kB
stations: 2640 kB
type_place: 1808 kB
guest_quantities: 6944 kB
zip_codes: 784 kB
districts: 64 kB
states: 24 kB
station_zip: 376 kB
```

Finally close the db connection

1.7 Conclusion:

- All tables could be set up and the auxiliary queries helped to discover and correct the errors in the data collection and preparation.
- the database is ready for use
- next the tables can be filled with the desired data (see next part)

1.7.1 Some personal reflections:

When creating the next database I should pay attention to a more consistent nomenclature of the tables and values. For example, the n:m table for the connection of `station` and `types` is

`type_place` instead of `station_types`. The determination of the actual storage space requirement can be made more precise.

In order to be able to share such a project with others, the database should be more easily portable; perhaps a SQLite database is better suited for this.