2η Εργασία στο μάθημα Big Data Mining

“Create your own, noisy dataset”

Γρηγόρης Μπουζιωτόπουλος

Βαγγέλης Κορμέντζας

Νικηφόρος Αλυγιζάκης

Διδάσκων: Ερευνητής Δρ. Η. Ζαβιτσάνος

Διϊδρυματικό πρόγραμμα μεταπτυχιακών σπουδών MSc in Data Science

Πανεπιστήμιο Πελοποννήσου και Εθνικό Κέντρο Έρευνας Φυσικών Επιστημών "ΔΗΜΟΚΡΙΤΟΣ"

Ημερομηνία κατάθεσης εργασίας: 27 Δεκεμβρίου 2020

Contents

Exercise in Big Data Mining
“Create your own, noisy dataset”

1. Description of the domain of interest

The field of interest is academia with focus on the faculty members and their activities. More specifically, we focused on the scientists, the faculty in which they work, their scientific projects, their supervision responsibilities, their participation in conferences and their scientific publications, including the announcements in conferences. The goal of our investigation was to create a relational database (11 relations in total), generate fake data, introduce noise to the dataset and propose interesting questions, requiring the retrieval of data from the database and the application of modeling.

2. Database schema

The database schema is outlined in Figure 1.

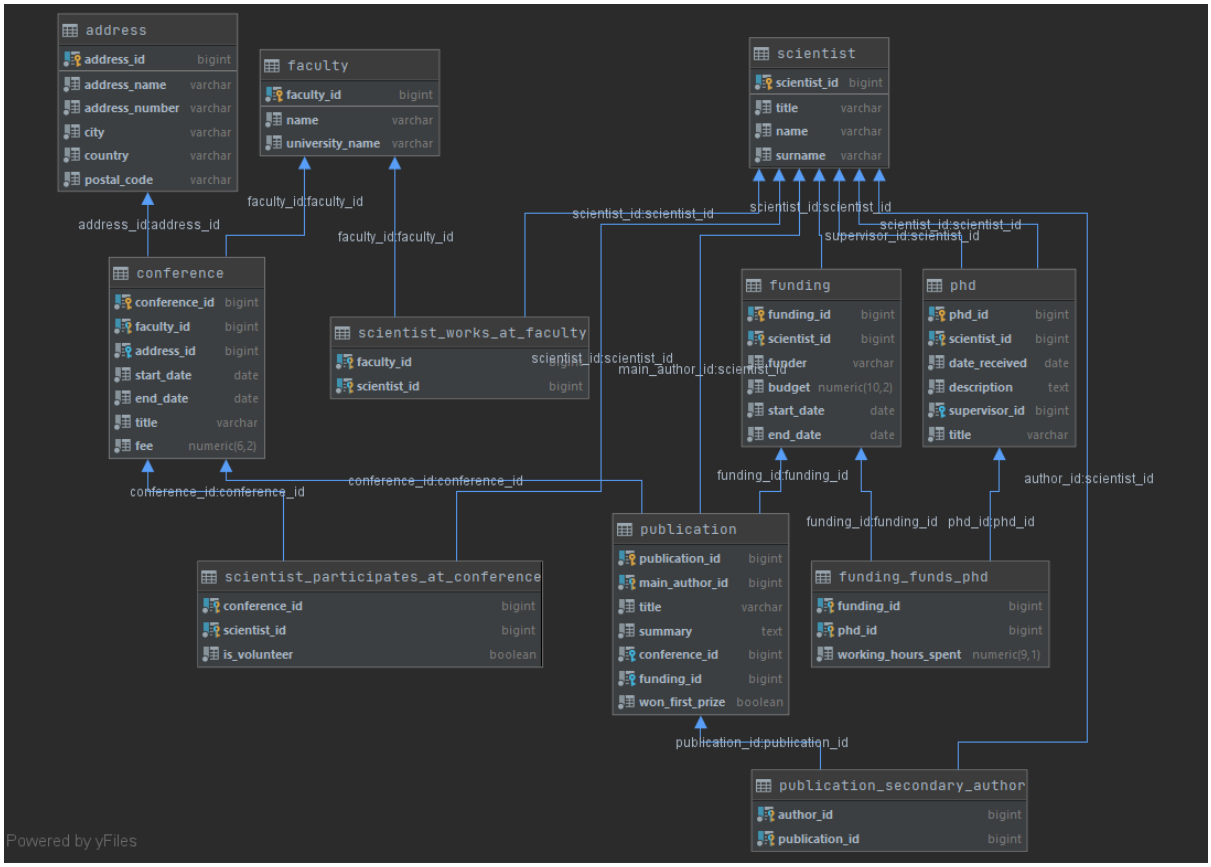

Figure 1: Database schema

2.1 Design practises and database overview:

The schema consists of 11 relations. For all the tables' primary keys, *big serial* data type was chosen in order to secure scalability. Since all pks are auto incremented, for all the foreign keys, no update and delete strategy was followed. Postal codes can contain special characters in some countries. Therefore, varchar was used. The same applies to address numbers, since they may contain the "-" character. Below is a description of the tables and the attributes for each table:

- **Table scientist**
  scientist_id: big serial -> primary key
  surname: varchar -> not null
  name: varchar -> not null
  title: varchar -> not null

- **Table faculty**
  faculty_id: big serial -> primary key
  faculty.name: varchar -> not null
  university_name: varchar -> not null

- **Table scientist_works_at_faculty**
  faculty_id : big int -> FK to table faculty(faculty_id), primary key
  scientist_id: big int -> FK to table scientist(scientist_id), primary key

- **Table phd**
  phd_id: big serial -> primary key
  date_received: date -> not null
  description: varchar -> not null
  supervisor_id: big int -> FK to table scientist(scientist_id)
  title: varchar -> not null
  scientist_id: big int -> FK to table scientist(scientist_id), primary key

- **Table address**
  address_id : big serial -> primary key
  address_name: varchar -> not null
  address_number: int -> not null
  city: varchar -> not null
  country: varchar -> not null
  postal_code: varchar -> not null

Comment: For addresses a separate entity was chosen, since many people can have the same address.

- **Table funding**
  funding_id : big serial -> primary key
  scientist_id: big int -> FK to table scientist(scientist_id), primary key
  funder: varchar -> not null
  budget: numeric(10,2) -> not null
  start_date: date -> not null

end_date: date -> not null

- **Table funding_funds_phd**
  funding_id: big int -> FK to table funding(funding_id), primary key
  phd_id: big int -> FK to table phd(phd_id), primary key
  working_hours_spent: numeric(9,2) -> not null

- **Table conference**
  conference_id: big serial -> primary key
  faculty_id: big int -> FK to table faculty(faculty_id), primary key
  address_id: big int -> FK to table address(address_id)
  start_date: date -> not null
  end_date: date -> not null
  title: varchar -> not null
  fee: numeric(6,2) -> not null

- **Table scientist_participates_at_conference**
  conference_id: big int -> FK to table conference(conference_id), primary key
  scientist_id: big int -> FK to table scientist(scientist_id), primary key
  is_voluteer: bool -> not null, default value: False

- **Table publication**
  publication_id: big serial -> primary key
  main_author_id: big int -> FK to table scientist(scientist_id), primary key
  title: varchar -> not null
  summary: text -> not null
  conference_id: big int -> FK to table conference(conference_id)
  funding_id: big int -> FK to table funding(funding_id)
  won_first_prize: bool -> not null, default value: False

- **Table publication_secondary_author**
  author_id: big int -> FK to table scientist(scientist_id), primary key
  publication_id: big int -> FK to table publication(publication_id), primary key

2.2 Using the schema

The schema resides in the *database* directory and provided that there is an active database, it can be loaded using the following command:

psql -f schema.sql <database_name>

The database name that we used while testing the scripts was “*scientific_community”*.

3.Strategy for filling in data and python code report

3.1 Prerequisites

The project was written in **Python 3.8.2.** It uses some new features of python 3.8. Therefore, it will work only with python **3.8** and above.

The package requirements can be found in the *requirements.txt* file (Figure 2). They can be installed to the python environment via the command:

pip install -r requirements.txt

```
Faker==5.0.2
pip==20.3.3
psycopg2-binary==2.8.6
python-dateutil==2.8.1
setuptools==51.1.0
six==1.15.0
text-unidecode==1.3
```

Figure 2: Contents of requirements.txt

A postgres database, should be already created with an **active schema** and **no data**. Details on how to create the schema can be found in *the Database Schema report above*

The structure of the python code is outlined in Figure 3.

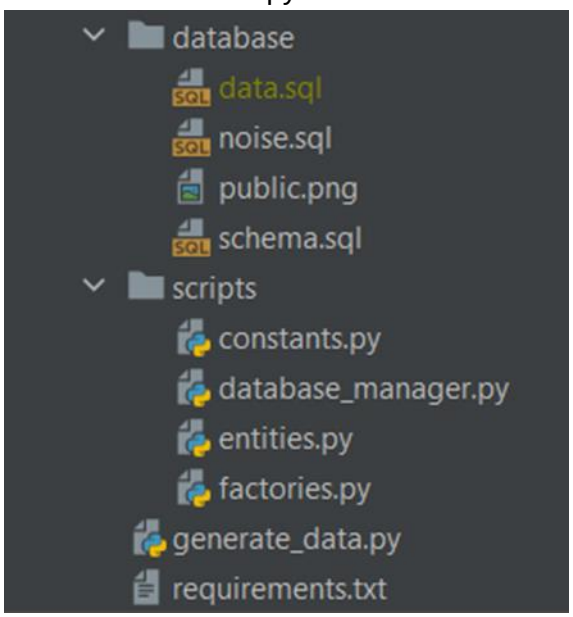

Figure 3: Directory structure of python code

3.2. Files

The **directory database** contains the 1) *schema.sql* file, used to generate the database schema, 2) the *public.png file*, which is the schematic of the database schema and 3) the noise.sql file used to introduce noise in the database. For convenience, the noisy data can also be found in 4) *data.sql*.

Notice**:** *data.sql* is not created by the script, but can be easily generated after executing the *generate_data.py* script, the *noise.sql* in order to introduce the noise in the data and finally using *pg_dump* to export the data

Example command to create the data.sql file:

pg_dump -d scientific_community -a --no-owner -f data.sql

The data can be inserted into the database (provided that it is empty and the schema has already been loaded) by the command below:

psql -f data.sql <database_name>

The **directory scripts** contains the following source code files:

entities.py: All database entities are represented as objects in this file, so that the data can be manipulated more efficiently. Figure 4, illustrates an example of the Conference entity.

```python
class Conference(BaseEntity):
    """Represents a Conference entity"""
    def __init__(self):
        self.conference_id = None
        self.faculty_id = None
        self.address_id = None
        self.start_date = None
        self.end_date = None
        self.title = None
        self.fee = None

    def __str__(self):
        return f"Conference(conference_id={self.conference_id}, faculty_id={self.faculty_id})"
```

Figure 4: Conference object in entities.py

factories.py: Utilizes the *faker* package in order to create entity objects with fake data.

FakeGenerator class is used in order to generate the data. Figure 5, illustrates an example of the AddressFactory.

```python
class AddressFactory(object):
    """Class used for generating fake Address entities"""

    @staticmethod
    def generate_addresses(n=1, start=1):
        """
        Generator of Address objects
        :param n: number of objects to be generated
        :param start: the starting id
        """
        for address_id in range(start, n + start):
            data = {"address_id": address_id, "address_name": _fg.address_name(),
                    "address_number": _fg.address_number(), "city": _fg.city(), "country": c.GREECE,
                    "postal_code": _fg.postal_code()}
            yield ent.Address.build_from_data(data)

    def __str__(self):
        return "AddressFactory"
```

Figure 5: UserFactory object in factory.py

constants.py: Contains all the constant variables that are used in the other scripts, such as the scientist titles etc. Figure 6 contains as example the scientist ranks.

```python
# Faculty staff titles
PROFESSOR = "Professor"
ASSOCIATE_PROFESSOR = "Associate Professor"
ASSISTANT_PROFESSOR = "Assistant Professor"
LECTURER = "Lecturer"
LABORATORY_TEACHING_STAFF = "Laboratory Teaching Staff"
RESEARCHER = "Researcher"
```

Figure 6: scientist titles in constants.py

database_manager.py: Contains the *ScientificCommunityDBManager* class, which handles all the connections to the database and also inserts the data into the database.

The algorithm used for data insertion is executed by the following method: *generate_and_insert_fake_data,* which does the following actions

1. Insert all faculties
2. Insert addresses proportional to the conference number
3. Insert conferences
4. Map conferences, addresses and faculties
5. For each faculty:
    a. create scientists
    b. split them into professors and non professors
    c. associate each one with the faculty
    d. create PhD for non professor scientists

e. assign PhD supervisors
f. create funding and associate them with professors
g. associate each PhD with a funding
h. create publications
i. map the publications to the conferences. Each one has a small chance to win a first prize in a conference.
j. create publications' secondary authors
k. assign scientists to conferences randomly

3.3 Execution instructions

The execution script is named ***generate_data.py***

Figure 7 illustrates the execution parameters required by the program. Each one is described by calling the *--help* option.

```
D:\PycharmProjects\data_mining_master_2020\venv\Scripts\python.exe D:/PycharmProjects/data_mining_master_2020/exercise_1/generate_data.py -h
usage: generate_data.py [-h] -d DATABASE -pwd PASSWORD [-u [USER]] [-i [IP]]
                        [-p [PORT]]

optional arguments:
  -h, --help            show this help message and exit
  -d DATABASE, --database DATABASE
                        the name of the database
  -pwd PASSWORD, --password PASSWORD
                        password for the specified database user
  -u [USER], --user [USER]
                        database user, defaults to postgres
  -i [IP], --ip [IP]    connection ip, defaults to localhost
  -p [PORT], --port [PORT]
                        connection port, defaults to 5432
```

Figure 7: Execution of generate_data.py

3.4. Execution examples:

1. *python generate_data.py -d scietific_community -pwd root*
2. *python generate_data.py -d scietific_community -pwd root -f main*
3. *python generate_data.py -d scietific_community -pwd root -f test*
4. *python generate_data.py -d scietific_community -pwd root -f test_rb*
5. *python generate_data.py -d scietific_community -u foo -ip 0.0.0.0 -p 1234 -pwd foobar*

After running the script, noise needs to be added by running the *noise.sql* file using the following command:

psql -f noise.sql <database_name>

4. Strategies for introducing noise and missing values

We followed the following insertion of noise in the database:

1. Duplicate records in the relation “fundings”. The same project is registered twice and received two *fundings_id*. We introduced noise in 3% of the records.
2. Duplicate records can also occur in the conferences (*conference* relation). We introduced noise in 3% of the records
3. Contradictory information: A professor is registered in two departments. We introduced noise of this type in 1 % of the records
4. Contradictory data in the name and surname of PhD students. For example, a name or surname is misspelled. We introduced noise in 10% of the names of the PhD students.
5. Missing data: PhD student is enrolled and the attribute *date_received* is contradictory. The secretary made mistakes while inserting the dates. 5% of the information is noisy.
6. Missing *start_date* and *funder* in relation *funding*. 3 % of the instances have either missing *start_date* or missing *funder*.
7. Missing *summary* or *title* in the relation *publication*. We introduced 5% of this noise in the respective relation.

The application of the noise strategies are applied in the file noise.sql

5. Interesting questions based on the dataset

1. Funding that the professors will raise in the upcoming years.
2. Scientific output (publications in peer-reviewed journals, announcements in conferences) that will be generated by the University in the upcoming years.
3. Identification of overperforming staff and PhD students (e.g., potential awards of excellence)
4. Number of PhD students needed to accomplish the scientific output of the projects