Bootcamp
Data Engineering

Day00
SQL with PostgreSQL

Bootcamp Data Engineering

Day00 - SQL with PostgreSQL

Today, you will learn how to use a SQL database: PostgreSQL.

Notions of the day

The purpose of the day is at first to create, administrate and normalize a PostgreSQL Database. Then, we are going to analyse the data and visualize the content of the database. Finally, we will see advanced notions like caching, replication and backups.

General rules

- The version of Python to use is 3.7, you can check the version of Python with the following command: `python -V`.
- For this day, you will follow the **Pep8 standard**.
- The exercises are ordered from the easiest to the hardest.
- Your exercises are going to be evaluated by someone else so make sure that variables and functions names are appropriated.
- Your man is the internet.
- You can also ask any question in the dedicated channel in Slack: **42ai slack**.
- If you find any issue or mistakes in the subject please create an issue on our dedicated repository on **Github issues**.

Foreword

Data Engineering implies many tasks from organizing the data to putting data systems to productions. Data organization is often a mess in companies and our job is to provide a common, well-organized data source. Historically, the organization of the data is used to analyze the business and determine future business decisions. Those data organizations are called Data warehouses and are used by business intelligence teams (teams in charge of analyzing the business). This organization of the data follows a star scheme allowing fast analysis.

Nowadays, we want to answer other cases like providing data to data science teams. To do that we want to provide a common data organization not specific to any project which will be used by anyone who wants it (business intelligence, data scientists, ...). This new data organization is called a Data Lake. It contains all the company data. The job of data engineering consists of organizing the data :

- ingestion
- storage
- catalog and search engine associated

To do that SQL is often used to filter, join, select the data. Today you will discover an open-source SQL language, PostgreSQL.

Helper

Ensure that you have the right Python version and psycopg2 installed.

```
$> which python
/goinfre/miniconda/bin/python
$> python -V
Python 3.7.*
$> which pip
/goinfre/miniconda/bin/pip
```

Exercise 00 - Setup

Exercise 01 - Clean

Exercise 02 - Normalize

Exercise 03 - Populate

Exercise 04 - Top_100

Exercise 05 - Name_lang

Exercise 06 - K-first

Exercise 07 - Seniors

Exercise 08 - Battle_royale

Exercise 09 - Benefits

Exercise 10 - Sweet_spot

Exercise 11 - Price_analysis

Exercise 12 - Worldwide

Exercise 13 - Sample

Exercise 00 - Setup

Turn-in directory :	ex00
Files to turn in :	None
Forbidden functions :	None
Remarks :	n/a

The client-server architecture

PostgreSQL is an open-source database wich follows a client-server architecture. It is divided in three different components :

- a **client**, a program on the user's machine which communicates the user's query to the server and receives the server's answers.
- a **server**, a program running in the background that manages access to a specific resource, service or network. The server will understand the client's query and apply it to the database. Then it will send an answer to the client.
- a **database system**, where the data is stored.

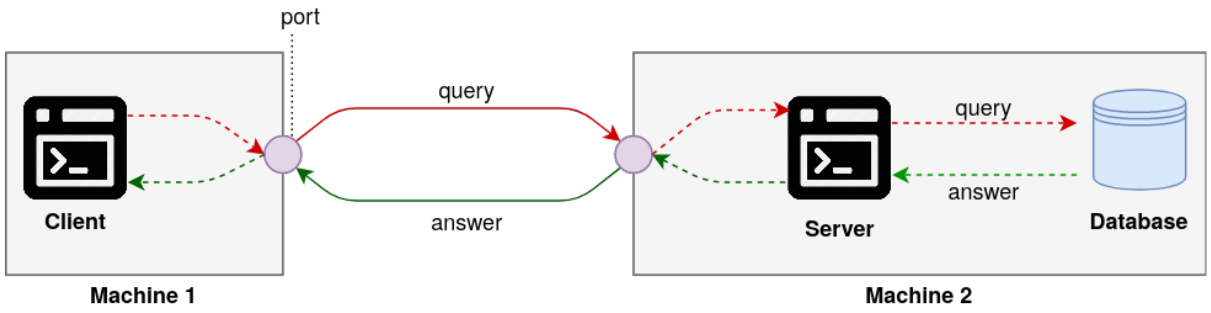

Figure 1: client-server architecture

ps: client and server can be located on the same machine

In the case of PostgreSQL, we are going to use `psql` as a client and `pg_ctl` for the server.

PostgreSQL dockerised architecture

Docker is a tool which allows us to isolate an environment like a virtual machine. HOWEVER, docker is not a virtual machine! It's more like an application running on the operating system and emulating an operating system.

An important point in using a database is the isolation and protection of the data. A way to do that is to isolate the server program from the database. We can use docker volumes combined with docker container to do that, as shown below.

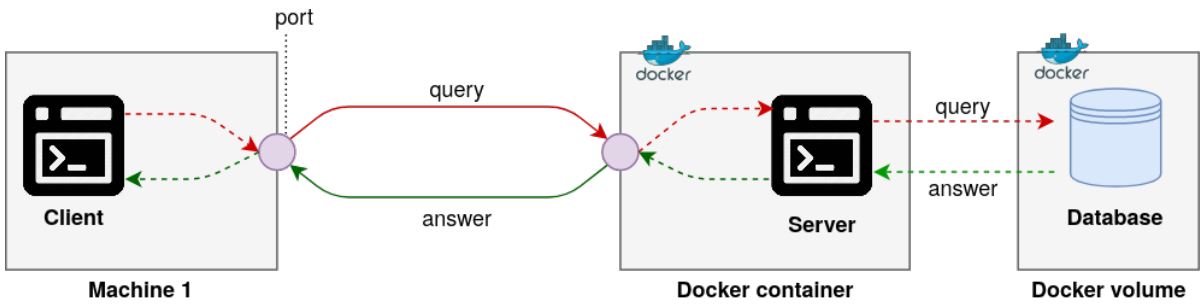

Figure 2: client-server architecture

The advantage of this architecture is that even if the docker container with the server fails, we still have the data on the docker volume.

PostgreSQL docker installation

We are going to install Postgres server through docker. The first thing we need to do is install Docker-CE. You can find a script to install docker on your mac in the `resource` directory of the day.

```
sh docker_install.sh
```

We just need a Postgres client now (`psql`).

```
brew install libpq
```

Then we need to create a docker volume. It will store the database data.

```
docker volume create db-data
```

Docker-compose will run a container with the last version of Postgres. The container Postgres port `5432` bound to the port `54320` of our local machine. The default user and password for Postgres is now `postgres`.

```
FROM postgres:12.2-alpine

# run init.sql
ADD init.sql /docker-entrypoint-initdb.d
ADD pg_hba.conf /var/lib/postgresql/data
```

The `init.sql` will create a user `postgres_user` with the password `12345` (a very strong password indeed!) and a database (`appstore_games`) for the day.

```
CREATE DATABASE appstore_games;
CREATE USER postgres_user WITH PASSWORD '12345';
ALTER DATABASE appstore_games OWNER TO postgres_user;
```

Now, you can run docker-compose!

```
$> docker-compose up
Starting resource_db_1 ...
Starting resource_db_1 ... done
Attaching to resource_db_1
db_1  |
db_1  | PostgreSQL Database directory appears to contain a database; Skipping initialization
db_1  |
db_1  | 2020-05-02 20:27:47.354 UTC [1] LOG:  starting PostgreSQL 12.2 on
 ↪  x86_64-pc-linux-musl, compiled by gcc (Alpine 9.2.0) 9.2.0, 64-bit
db_1  | 2020-05-02 20:27:47.354 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
db_1  | 2020-05-02 20:27:47.354 UTC [1] LOG:  listening on IPv6 address "::", port 5432
db_1  | 2020-05-02 20:27:47.359 UTC [1] LOG:  listening on Unix socket
 ↪  "/var/run/postgresql/.s.PGSQL.5432"
db_1  | 2020-05-02 20:27:47.370 UTC [20] LOG:  database system was shut down at 2020-05-02
 ↪  20:23:08 UTC
db_1  | 2020-05-02 20:27:47.373 UTC [1] LOG:  database system is ready to accept connections
```

Now, we can connect to the server through `psql`. As said previously, the password is `postgres`.

```
$> psql -h 0.0.0.0 -p 54320 -U postgres
psql (12.1)
Type "help" for help.

postgres=#
```

`\?` allows you to see all the possible commands in the PostgreSQL console. The first thing we can do is list the databases with `\l`.

```
postgres=# \l
List of databases
Name      |     Owner     | Encoding |  Collate   |   Ctype    |   Access privileges
----------------+---------------+----------+------------+------------+-----------------------
appstore_games | postgres_user | UTF8     | en_US.utf8 | en_US.utf8 |
postgres       | postgres      | UTF8     | en_US.utf8 | en_US.utf8 |
template0      | postgres      | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
|              |          |            |            | postgres=CTc/postgres
template1      | postgres      | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
|              |          |            |            | postgres=CTc/postgres
```

We are ready to use Postgres!

Pyenv install

Dealing with Python is often hell when it comes to python versions and librairies version. This problem is often encountered few people are working on the same server with different library needs. Furthermore you don't want to mess with the system python. That's why virtual environments and separated python are a prefered solution.

You can install pyenv with brew using the following command.

```
brew install pyenv
```

All the python candidates can then be listed.

```
pyenv install --list | grep " 3\.[678]"
```

... and installed. For the day we are going to choose version `3.8.0`.

```
pyenv install -v 3.8.0
```

Finally the installed version can be activated through this command.

```
pyenv global 3.8.0
```

Don't forget to add those lines to your `.zshrc` file in order to activate your python environment each time you open a terminal.

```
export PATH="/home/misteir/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"

pyenv global 3.8.0 #activate the python 3.8.0 as default python
```

Pipenv install

Pipenv is a tool to handle packages versions of an environment. This tool is very similar to the `requirements.txt` file with some extra metadata.

Pipenv can be installed with this simple command.

```
pip install pipenv
```

You can find a toml file for the day named `Pipfile`.

```
[[source]]
url = "https://pypi.python.org/simple"
verify_ssl = true
name = "pypi"

[packages]
jupyter = "*"
numpy = "*"
pandas = "*"
psycopg2 = "*"

[requires]
python_version = "3.8.0"
```

To setup your environment just follow these two steps.

```
pipenv install
pipenv shell
```

You have now PostgreSQL, virtual python and requirements installed and ready for the day!

Exercise 01 - Clean

Turnin directory :	ex01
Files to turn in :	clean.py
Forbidden function :	None
Remarks :	n/a

Objective

You must clean the given CSV dataset to insert it into a PostgreSQL table.

Instructions

The `appstore_games.csv` file is available in the resources.

We are going to keep the following columns: `ID`, `Name`, `Average User Rating`, `User Rating Count`, `Price`, `Description`, `Developer`, `Age Rating`, `Languages`, `Size`, `Primary Genre`, `Genres`, `Original Release Date`, `Current Version Release Date`.

1) You need to implement the function `df_nan_filter`. It takes a pandas dataframe as input and applies the following replacement for NaN values :

- remove the row if `Size` if NaN.
- set `Languages` as "EN" if NaN.
- set `Price` as 0.0 if NaN.
- set `Average User Rating` as the median of the column if NaN.
- set `User Rating Count` as 1 if NaN.

```
def df_nan_filter(df):
"""Apply filters on NaN values
Args:
df: pandas dataframe.
Returns:
Filtered Dataframe.
Raises:
This function shouldn't raise any Exception.
"""
```

1) Create the function `change_date_format` that will change the date format from `dd/mm/yyyy` to `yyyy-mm-dd`.

```
def change_date_format(date: str):
"""Change date format from dd/mm/yyyy to yyyy-mm-dd
Args:
date: a string representing the date.
Returns:
The date in the format yyyy-mm-dd.
Raises:
This function shouldn't raise any Exception.
"""
```

Your function must work with the following commands.

```
df["Original Release Date"] = df["Original Release Date"].apply(lambda x:
↪  change_date_format(x))
df["Current Version Release Date"] = df["Current Version Release Date"].apply(lambda x:
↪  change_date_format(x))
```

3) You need to apply the following function to the `Description` column.

```
import re

def string_filter(s: str):
"""Apply filters in order to clean the string.
Args:
s: string.
Returns:
Filtered String.
Raises:
This function shouldn't raise any Exception.
"""
# filter : \\t, \\n, \\U1a1b2c3d4, \\u1a2b, \\x1a
# turn \' into '
# replace remaining \\ with \
# turn multiple spaces into one space
s = re.sub(r'''\\+(t|n|U[a-z0-9]{8}|u[a-z0-9]{4}|x[a-z0-9]{2}|[\.]{2})''', ' ', s)
s = s.replace('\\\'', '\'').replace('\\\\', '\\')
s = re.sub(r' +', ' ', s)
return (s)
```

4) Remove the `ID` duplicates.

5) Lastly, convert the data type of the columns `Age Rating`, `User Rating Count` and `Size` to int.

You must apply these 4 steps to create a script producing the file `appstore_games.cleaned.csv`.

Examples

The following example does not show the true dataset and values obtained after the filters.

```
>>> df = pd.read_csv("appstore_games.csv")
>>> df.head(1)
Average User Rating  User Rating Count  Price Languages
1                   NaN                NaN    NaN        NaN
>>> df = nan_filter(df)
>>> df.head(1)
Age User Rating  User Rating Count  Price Languages
4                   1     15        EN
```

```
for e in df:
print("'{}' :: {}".format(e, df.loc[0, e]))
```

With the above code, you should obtain something similar to this output for the values of the first row. The output shape is (16846, 14).

```
'ID' :: 284921427
'Name' :: Sudoku
'Average User Rating' :: 4.0
'User Rating Count' :: 3553
'Price' :: 2.99
'Description' :: Join over 21,000,000 of our fans and download one of our Sudoku games
↪  today! Makers of the Best Sudoku Game of 2008, Sudoku (Free), we offer you the best
↪  selling Sudoku game for iPhone with great features and 1000 unique puzzles! Sudoku will
↪  give you many hours of fun and puzzle solving. Enjoy the challenge of solving Sudoku
↪  puzzles whenever or wherever you are using your iPhone or iPod Touch. OPTIONS All
↪  options are on by default, but you can turn them off in the Options menu Show Incorrect
↪  :: Shows incorrect answers in red. Smart Buttons :: Disables the number button when that
↪  number is completed on the game board. Smart Notes :: Removes the number from the notes
↪  in the box, column, and row that contains the cell with your correct answer. FEATURES
↪  1000 unique handcrafted puzzles ALL puzzles solvable WITHOUT guessing Four different
↪  skill levels Challenge a friend Multiple color schemes ALL notes: tap the All notes
↪  button on to show all the possible answers for each square. Tap the All notes button off
↪  to remove the notes. Hints: shows the answer for the selected square or a random square
↪  when one is not selected Pause the game at any time and resume where you left off Best
↪  times, progress statistics, and much more Do you want more? Try one of our other
↪  versions of sudoku which have all the same great features! * Try Color Sudoku for a fun
↪  twist to solving sudoku puzzles. * For advanced puzzle solving, try Expert Sudoku to
↪  challenge your sudoku solving skills.
'Developer' :: Mighty Mighty Good Games
'Age Rating' :: 4
'Languages' :: DA, NL, EN, FI, FR, DE, IT, JA, KO, NB, PL, PT, RU, ZH, ES, SV, ZH
'Size' :: 15853568
'Primary Genre' :: Games
'Genres' :: Games, Strategy, Puzzle
'Original Release Date' :: 2008-07-11
'Current Version Release Date' :: 2017-05-30
```

Exercise 02 - Normalize

Turnin directory :	ex02
Files to turn in :	normalize.py
Forbidden function :	None
Remarks :	n/a

Objective

You must normalize the given CSV dataset to insert it into a PostgreSQL table.

Instructions

We are going to use the previously cleaned dataset and apply the `1NF normalization` rule to it.

1NF normalization

- Each column should contain atomic values (list entries like `x, y` violate this rule).
- Each column should contain values of the same type.
- Each column should have unique names.
- Order in which data is saved does not matter.

This rule is normally applied to a database but we are going to use those data as database tables in the next exercises.

The only rule that we are not following concerns the list of values in columns. Not respecting this rule will complicate queries a lot (querying on a list is not convenient).

The two columns that don't respect this rule are `Languages` and `Genres`. In order to respect the 1NF rule you have to create 3 dataframes (that are going to be postgresql tables) :

- **df** : `ID`, `Name`, `Average User Rating`,`User Rating Count`, `Price`, `Description`, `Developer`, `Age Rating`, `Size`, `Original Release Date`, `Current Version Release Date`
- **df_genres** : `ID`, `Primary Genre`, `Genre`
- **df_languages** : `ID`, `Language`

We want to go from this form ...

```
ID         Languages
0  284921427        DA, NL, EN
```

... to this one.

```
ID Language
0  284921427       DA
1  284921427       NL
2  284921427       EN
```

To do that we can use the `explode` function of pandas. This function only works with lists so we have to convert the string `DA, NL, EN` to a list format like `[DA, NL, EN]`.

1) Create the 3 dataframes (with the corresponding columns)
2) Convert multiple words genres to a single word format (ex: `Arcade & Aventure` to `Arcade_&_Aventure`)
3) Convert strings to list format (for columns with list) and remove the 'Games' genre from each list (it is irrelevant information as it is in each list)

4) Use the `explode` function of pandas (index of dataframes will be broken)

5) reset the index of the dataframes (`reset_index` function)

6) Save the dataframes into the files :

- `appstore_games.normalized.csv` (shape : (16846, 11))
- `appstore_games_genres.normalized.csv` (shape : (44351, 3))
- `appstore_games_languages.normalized.csv` (shape : (54823, 2))

Examples

```
>> print(df_languages.head())
ID Language
0  284921427       DA
1  284921427       NL
2  284921427       EN
3  284921427       FI
4  284921427       FR
```

```
>> print(df_genres.head())
ID Primary Genre     Genre
0  284921427         Games  Strategy
1  284921427         Games    Puzzle
2  284926400         Games  Strategy
3  284926400         Games     Board
4  284946595         Games     Board
```

Exercise 03 - Populate

Turnin directory :	ex03
Files to turn in :	populate.py
Forbidden function :	None
Remarks :	n/a

Objective

You must insert :

- `appstore_games.normalized.csv`
- `appstore_games_genres.normalized.csv`
- `appstore_games_languages.normalized.csv`

data into a PostgreSQL table.

Instructions

You can read the psycopg2_basics documentation (some included functions will help you with this exercise).

1) You first need to create 3 functions.

- `create_appstore_games`
- `create_appstore_games_genres`
- `create_appstore_games_languages`

... to create the following tables :

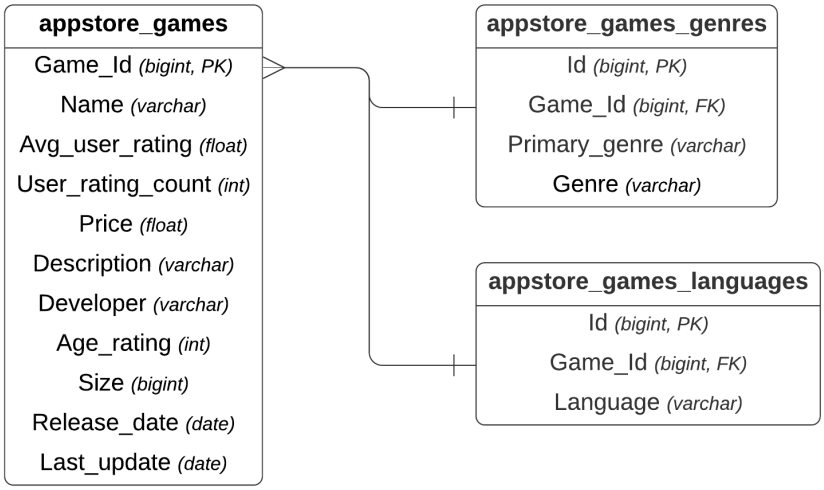

Figure 3: tables

nb: Foreign keys are a reference to an existing column in another table.

2) You will have to create the 3 populate functions

- `populate_appstore_games`
- `populate_appstore_games_genres`

- `populate_appstore_games_languages`

... to insert data into the different tables.

Before you do anything you must ensure postgresql is running.

Examples

At the end your display table should show the following output for the table :

- `appstore_games_genres`

```
>> display_table("appstore_games_genres")
(0, 284921427, 'Games', 'Strategy')
(1, 284921427, 'Games', 'Puzzle')
(2, 284926400, 'Games', 'Strategy')
(3, 284926400, 'Games', 'Board')
(4, 284946595, 'Games', 'Board')
(5, 284946595, 'Games', 'Strategy')
(6, 285755462, 'Games', 'Strategy')
(7, 285755462, 'Games', 'Puzzle')
(8, 285831220, 'Games', 'Strategy')
(9, 285831220, 'Games', 'Board')
```

- `appstore_games_languages`

```
>> display_table("appstore_games_languages")
(0, 284921427, 'DA')
(1, 284921427, 'NL')
(2, 284921427, 'EN')
(3, 284921427, 'FI')
(4, 284921427, 'FR')
(5, 284921427, 'DE')
(6, 284921427, 'IT')
(7, 284921427, 'JA')
(8, 284921427, 'KO')
(9, 284921427, 'NB')
```

Exercise 04 - Top100

Turnin directory :	ex04
Files to turn in :	top100.py
Forbidden functions :	None
Remarks :	n/a

Objective

You must show the top 100 games Name with the best user rating.

Instructions

You must create a program using the function `get_top_100`.

This function must show the top 100 games Name ordered by `Avg_user_rating` first then by `Name`.

The names of games not starting with a letter must be ignored. Then, you must show the first 100 games starting with letters.

You must only use PostgreSQL for your queries !

Example

```
>> get_top_100()
AFK Arena
APORIA
AbsoluteShell
Action Craft Mini Blockheads Match 3 Skins Survival Game
Adrift by Tack
Agadmator Chess Clock
Age Of Magic
Age of Giants: Tribal Warlords
Age of War Empires: Order Rise
Alicia Quatermain 2 (Platinum)
...
```

Exercise 05 - Name_lang

Turnin directory :	ex05
Files to turn in :	name_lang.py
Forbidden functions :	None
Remarks :	n/a

Objective

You must show Name and Language of games strictly between 5 and 10 euros both excluded.

Instructions

You must create a program using the function `get_name_lang` that will show the Name and Language of games strictly between 5 and 10 euros.

You must only use PostgreSQL for your queries !

Example

```
>> get_name_lang()
Chess Genius EN
Chess Genius FR
Chess Genius DE
Chess Genius IT
Chess Genius ES
Chess - tChess Pro EN
Chess - tChess Pro FR
Chess - tChess Pro DE
Chess - tChess Pro JA
Chess - tChess Pro KO
Chess - tChess Pro ZH
...
```

Exercise 06 - K-first

Turnin directory :	ex06
Files to turn in :	k_first.py
Forbidden functions :	None
Remarks :	n/a

Objective

You must show the name of developers starting with 'K' and involved in casual games.

Instructions

You must create a program using the function `get_k_first` that shows the name of developers starting with 'K' (case sensitive) and involved in casual games.

You must only use PostgreSQL for your queries !

Example

```
>> get_k_first()
Koh Jing Yu
Kyle Decot
Kashif Tasneem
Kristin Nutting
Kok Leong Tan
Key Player Publishing Limited
KillerBytes
KillerBytes
Khoa Tran
Kwai Ying Cindy Cheung
KG2 Entertainment LLC
Keehan Roberts
...
```

Exercise 07 - Seniors

Turnin directory :	ex07
Files to turn in :	seniors.py
Forbidden functions :	None
Remarks :	n/a

Objective

You must show the Name of developers involved in games released before 01/08/2008 included and updated after 01/01/2018 included.

Instructions

You must create a program using a function `get_seniors` that shows the Name of developers involved in games released before 01/08/2008 included and updated after 01/01/2018 included.

You must only use PostgreSQL for your queries !

Example

```
>> get_seniors()
Kiss The Machine
...
```

Exercise 08 - Battle_royale

Turnin directory :	ex08
Files to turn in :	battle_royale.py
Forbidden functions :	None
Remarks :	n/a

Objective

You must show the name of the games with "battle royale" in their description and with a URL that will redirect to `facebook.com`.

Instructions

You must create a program using a function `get_battle_royale` that shows the name of the games with "battle royale" (case insensitive) in their description and with a URL that will redirect to `facebook.com`.

You must only use PostgreSQL for your queries !

Example

```
>> get_battle_royale()
Lords Mobile: War Kingdom
Crusaders of Light
Blob io - Throw & split cells
...
```

Exercise 09 - Benefits

Turnin directory :	ex09
Files to turn in :	benefits.py
Forbidden function :	None
Remarks :	n/a

Objective

Show the first 10 games that generated the most benefits.

Instructions

You must create a program using the function `get_benefits` that will show the first 10 games that generated the most benefits.

Benefits are calculated with the number of users who voted times the price of the game.

You must only use PostgreSQL for your queries !

Example

```
>> get_benefits()
Bloons TD 5
Bloons TD 6
Traffic Rush
Plague Inc.
Chess Tiger Pro
Finabase: realtime stocks
Bloons TD 5 HD
...
```

Exercise 10 - Sweet spot

Turnin directory :	ex10
Files to turn in :	sweet_spot.py
Forbidden function :	None
Remarks :	n/a

Objective

Find the month where the most important number of games are released.

Instructions

Find the month where the most important number of games are released.

You must only use PostgreSQL for your queries !

Example

This answer may not be the right one.

```
january
```

Exercise 11 - Price Analysis

Turnin directory :	ex11
Files to turn in :	price.py, price.png
Forbidden function :	None
Remarks :	n/a
Allowed python libraries :	matplotlib, numpy

Objective

Analyze the price distribution of games by plotting a histogram of the price distribution.

Instructions

First, you need to write the right query to output a table where you have the distribution of price, i.e. the number of games for each price.

Then, you can use matplotlib to create a histogram. Your histogram will have to :

- not show games with a price below 1.0
- have a bar plot with 3 euros interval
- have the xlabel `Price`
- have the ylabel `Frequency`
- have the title `Appstore games price`

You will have to save your histogram in a file named `price.png`

Finally, you have to use numpy to find the mean and the standard deviation of your data set.

nb: you do not need to worry about the number of decimals printed

You can use PostgreSQL and Python (for numpy, matplotlib, bins creation ...)

Example

This answer may not be the right one.

```
> python price.py
mean price :  15.04098
std price :  6.03456
```

Exercise 12 - Worldwide

Turnin directory :	ex12
Files to turn in :	worldwide.py
Forbidden function :	None
Remarks :	n/a

Objective

Give the top 5 most played genres among games that have several distinct languages greater or equal to 3.

Instructions

You must write a query that filters games according to the number of languages they have, and then filter out the ones that have strictly less than 3 languages. Then you need to select the top 5 genres where those games appear.

You must only use PostgreSQL for your queries !

Example

```
$> python worldwide.py
Strategy
...
```

Exercise 13 - Sample

Turnin directory :	ex13
Files to turn in :	sample.py
Forbidden function :	None
Remarks :	n/a

Objective

Create a statistically representative sample of your dataset.

Instructions

1) we need to find a good sample size for our dataset. You can find information related to representative sample size calculation at the following **wiki**, **wikihow**.

When you understood how it works, find a sample size calculator online and compute the sample size using the given parameters :

- The margin of error of 5%
- Confidence Level of 95%
- population size (size of appstore_games table)

Then put the sample size in a variable.

2) Write a PostgreSQL `sample` function that will randomly select a given number of rows (sample_size parameter)

3) Use your `sample` function to randomly select a sample and save the result into a CSV file named `appstore_games.sample.csv`

Hint : you can use `pd.read_sql_query` and `df.to_csv` !

You must only use PostgreSQL for your queries !

Bonus

Write a python function `sample_size` with the following parameters :

- `population_size`
- `confidence_level` : default value `0.95`
- `margin_error` : default value `0.05`
- `standard_deviation` : default value `0.5`

This function will compute the sample size needed for the given parameter following the given formula :

$$sample\_size = \frac{\frac{zscore^2 \times std(1-std)}{margin\_error^2}}{1 + \frac{zscore^2 \times std(1-std)}{margin\_error^2 \times Population\_size}}$$

The z_score depends on the confidence level following this table:

Confidence_level	Z_score
0.80	1.28
0.85	1.44
0.90	1.65
0.95	1.96
0.99	2.58

You can follow **this guide** if you want to understand more about each parameter.