Advanced Databases

Project description

Mateusz Mazurkiewicz

Kamil Paśko

1. Dataset

In our project we used dataset from link. It contains data about game sales for different years. The database consists of 10 columns: *Name, Year, Genre, Publisher, NA_Sales, EU_Sales, JP_Sales, Other_Sales, Global_Sales.* There are almost 16600 records in the database.

	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	Other_Sales	Global_Sales
Rank										
1	Wii Sports	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	8.46	82.74
2	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	0.77	40.24
3	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	3.31	35.82
4	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.75	11.01	3.28	2.96	33.00
5	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	1.00	31.37
...	...	...	...	...	...	...	...	...	...	...
16596	Woody Woodpecker in Crazy Castle 5	GBA	2002.0	Platform	Kemco	0.01	0.00	0.00	0.00	0.01
16597	Men in Black II: Alien Escape	GC	2003.0	Shooter	Infogrames	0.01	0.00	0.00	0.00	0.01
16598	SCORE International Baja 1000: The Official Game	PS2	2008.0	Racing	Activision	0.00	0.00	0.00	0.00	0.01
16599	Know How 2	DS	2010.0	Puzzle	7G//AMES	0.00	0.01	0.00	0.00	0.01
16600	Spirits & Spells	GBA	2003.0	Platform	Wanadoo	0.01	0.00	0.00	0.00	0.01

2. Code

We start with reading our dataset from csv.

```
import pandas as pd


game_sales_data = pd.read_csv('vgsales.csv', index_col=0)
game_sales_data
```

Then we import necessary dependencies and establish database connection.

```
from sqlalchemy import Column, Integer, String, Date, Text, VARCHAR, Float, MetaData, Table, select
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import ForeignKey
from sqlalchemy import CheckConstraint, UniqueConstraint
from sqlalchemy.orm import sessionmaker


Base = declarative_base()

from sqlalchemy import create_engine

db_string = "postgres://postgres:admin@localhost:5432/game_sales"
engine = create_engine(db_string)
```

In the next step we define classes to facilitate object mapping. After that we create list or two dimensional list for each column.

```
platforms_list = pd.DataFrame(columns=['platform_name'])
platforms_list['platform_name'] = game_sales_data['Platform'].unique()
platforms_list.index.name = 'platform_id'
platforms_list
```

```
categories_list = pd.DataFrame(columns=['category_name'])
categories_list['category_name'] = game_sales_data['Genre'].unique()
categories_list.index.name = 'category_id'
categories_list
```

```
publishers_list = pd.DataFrame(columns=['publisher_name'])
publishers_list['publisher_name'] = game_sales_data['Publisher'].unique()
publishers_list.index.name = 'publisher_id'
publishers_list
```

```
games_list = game_sales_data[['Name', 'Platform', 'Year', 'Genre', 'Publisher']]

def map_name_to_id(name, column_name, names_list):
    result = names_list[names_list[column_name] == name].index.values.astype(int)
    if len(result) > 0:
        return result[0]
    else:
        return None

games_list.index.name = 'game_id'
games_list = games_list.rename(columns = {'Name':'game_name', 'Platform': 'platform_id', 'Year': 'year', 'Genre': 'category_id', 'Publish
er': 'publisher_id'})
games_list['platform_id'] = games_list['platform_id'].map(lambda x: map_name_to_id(x, 'platform_name', platforms_list))
games_list['publisher_id'] = games_list['publisher_id'].map(lambda x: map_name_to_id(x, 'publisher_name', publishers_list))
games_list['category_id'] = games_list['category_id'].map(lambda x: map_name_to_id(x, 'category_name', categories_list))
games_list
```

```
sales_list = game_sales_data[['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales']]
sales_list
```

Then we insert our new lists as columns to database.

```
platforms_list.to_sql('Platform', engine, if_exists='append')
categories_list.to_sql('Category', engine, if_exists='append')
publishers_list.to_sql('Publisher', engine, if_exists='append')
games_list.to_sql('Game', engine, if_exists='append')
sales_list.to_sql('Sales', engine, if_exists='append')
```

For validation purposes we create exemplary SELECT function.

```
mapper_stmt = select([dic_table['Platform'].columns.platform_id,dic_table['Platform'].columns.platform_name])
print('Mapper select: ')
print(mapper_stmt)
mapper_results = engine.execute(mapper_stmt).fetchall()
print(mapper_results)
print('\nNumber of platforms:', len(mapper_results))
```

```
Mapper select:
SELECT "Platform".platform_id, "Platform".platform_name
FROM "Platform"
[(0, 'Wii'), (1, 'NES'), (2, 'GB'), (3, 'DS'), (4, 'X360'), (5, 'PS3'), (6, 'PS2'), (7, 'SNES'), (8, 'GBA'), (9, '3DS'), (10, 'PS4'), (1
1, 'N64'), (12, 'PS'), (13, 'XB'), (14, 'PC'), (15, '2600'), (16, 'PSP'), (17, 'XOne'), (18, 'GC'), (19, 'WiiU'), (20, 'GEN'), (21, 'D
C'), (22, 'PSV'), (23, 'SAT'), (24, 'SCD'), (25, 'WS'), (26, 'NG'), (27, 'TG16'), (28, '3DO'), (29, 'GG'), (30, 'PCFX')]

Number of platforms: 31
```

Next we fetch some interesting data:

- 30 Wii games

```
# 30 Wii games
mapper_stmt = select([dic_table['Game'].columns.game_name]).\
    where(dic_table['Game'].columns.platform_id.in_(select([dic_table['Platform'].columns.platform_id]).\
        where(dic_table['Platform'].columns.platform_name == 'Wii'))).limit(30)
print('Mapper select: ')
print(mapper_stmt)
mapper_results = engine.execute(mapper_stmt).fetchall()
print(mapper_results)
```

```
Mapper select:
SELECT "Game".game_name
FROM "Game"
WHERE "Game".platform_id IN (SELECT "Platform".platform_id
FROM "Platform"
WHERE "Platform".platform_name = :platform_name_1)
 LIMIT :param_1
[('Wii Sports',), ('Mario Kart Wii',), ('Wii Sports Resort',), ('Wii Play',), ('New Super Mario Bros. Wii',), ('Wii Fit',), ('Wii Fit Plu
s',), ('Super Smash Bros. Brawl',), ('Super Mario Galaxy',), ('Just Dance 3',), ('Just Dance 2',), ('Wii Party',), ('Mario Party 8',),
('Mario & Sonic at the Olympic Games',), ('Super Mario Galaxy 2',), ('The Legend of Zelda: Twilight Princess',), ('Just Dance',), ('Just
Dance 4',), ('Zumba Fitness',), ('Donkey Kong Country Returns',), ('LEGO Star Wars: The Complete Saga',), ("Link's Crossbow Training",),
('Animal Crossing: City Folk',), ('Guitar Hero III: Legends of Rock',), ('Mario & Sonic at the Olympic Winter Games',), ('Michael Jackso
n: The Experience',), ('The Legend of Zelda: Skyward Sword',), ('Carnival Games',), ('EA Sports Active',), ('Big Brain Academy: Wii Degre
e',)]
```

- 10 Best selling games

```
# 10 Best selling games
mapper_stmt = select([dic_table['Game'].columns.game_name, dic_table["Sales"].columns.Global_Sales]).\
    where(dic_table['Game'].columns.game_id == dic_table["Sales"].columns.game_id).\
    order_by(dic_table["Sales"].columns.game_id).limit(10)
print('Mapper select: ')
print(mapper_stmt)
mapper_results = engine.execute(mapper_stmt).fetchall()
print(mapper_results)
```

```
Mapper select:
SELECT "Game".game_name, "Sales"."Global_Sales"
FROM "Game", "Sales"
WHERE "Game".game_id = "Sales".game_id ORDER BY "Sales".game_id
 LIMIT :param_1
[('Wii Sports', 82.74), ('Super Mario Bros.', 40.24), ('Mario Kart Wii', 35.82), ('Wii Sports Resort', 33.0), ('Pokemon Red/Pokemon Blu
e', 31.37), ('Tetris', 30.26), ('New Super Mario Bros.', 30.01), ('Wii Play', 29.02), ('New Super Mario Bros. Wii', 28.62), ('Duck Hunt',
28.31)]
```

- 10 Worst selling games

```
# 10 Worst selling games
mapper_stmt = select([dic_table['Game'].columns.game_name, dic_table["Sales"].columns.Global_Sales]).\
    where(dic_table['Game'].columns.game_id == dic_table["Sales"].columns.game_id).\
    order_by(dic_table["Sales"].columns.game_id.desc()).limit(10)
print('Mapper select: ')
print(mapper_stmt)
mapper_results = engine.execute(mapper_stmt).fetchall()
print(mapper_results)
```

```
Mapper select:
SELECT "Game".game_name, "Sales"."Global_Sales"
FROM "Game", "Sales"
WHERE "Game".game_id = "Sales".game_id ORDER BY "Sales".game_id DESC
 LIMIT :param_1
[('Spirits & Spells', 0.01), ('Know How 2', 0.01), ('SCORE International Baja 1000: The Official Game', 0.01), ('Men in Black II: Alien E
scape', 0.01), ('Woody Woodpecker in Crazy Castle 5', 0.01), ('Plushees', 0.01), ('Myst IV: Revelation', 0.01), ('Eiyuu Densetsu: Sora no
Kiseki Material Collection Portable', 0.01), ('Chou Ezaru wa Akai Hana: Koi wa Tsuki ni Shirube Kareru', 0.01), ('Mega Brain Boost', 0.0
1)]
```

We define two additional functions and check if they work correctly.

- get_games_for_plat- returns all games for given platform

```
#function finding all games for given platform
mapper_stmt = '''CREATE OR REPLACE FUNCTION get_games_for_plat(plat_name VARCHAR)\
RETURNS TABLE (game_name VARCHAR) AS $$ \
BEGIN RETURN QUERY SELECT "Game".game_name FROM "Game" INNER JOIN "Platform" on "Game".platform_id="Platform".platform_id \
WHERE "Platform".platform_name = plat_name;\
END;$$ \
LANGUAGE 'plpgsql';'''

print(mapper_stmt)
mapper_results = engine.execute(mapper_stmt)
```

```
CREATE OR REPLACE FUNCTION get_games_for_plat(plat_name VARCHAR)RETURNS TABLE (game_name VARCHAR) AS $$ BEGIN RETURN QUERY SELECT "Game".
game_name FROM "Game" INNER JOIN "Platform" on "Game".platform_id="Platform".platform_id WHERE "Platform".platform_name = plat_name;EN
D;$$ LANGUAGE 'plpgsql';
```

```
mapper_stmt = "Select * from get_games_for_plat('PC') limit 30"
print(mapper_stmt)
mapper_results = engine.execute(mapper_stmt).fetchall()
print(mapper_results)
```

```
Select * from get_games_for_plat('PC') limit 30
[('The Sims 3',), ('World of Warcraft',), ('Diablo III',), ('Microsoft Flight Simulator',), ('StarCraft II: Wings of Liberty',), ('Warcra
ft II: Tides of Darkness',), ('Half-Life',), ('World of Warcraft: The Burning Crusade',), ('The Elder Scrolls V: Skyrim',), ('The Sims: U
nleashed',), ('Doom II: Hell on Earth',), ('The Sims: Vacation',), ('The Sims: Livin Large',), ('The Sims 4',), ('Star Wars: The Old Repu
blic',), ('Command & Conquer: Red Alert',), ('Myst',), ('Battlefield 3',), ('Riven: The Sequel to Myst',), ('Theme Hospital',), ('Monopol
y',), ('Half-Life 2',), ('Guild Wars 2',), ('Tomb Raider II',), ('The Sims: House Party',), ('SimCity 2000',), ('World of Warcraft: Catac
lysm',), ('Sim Theme Park',), ('Warcraft: Orcs & Humans',), ('Star Wars: Dark Forces',)]
```

- get_games_for_cat- returns all games for given category

```
#function finding all games of given category
mapper_stmt = '''CREATE OR REPLACE FUNCTION get_games_for_cat(cat_name VARCHAR)\
RETURNS TABLE (game_name VARCHAR) AS $$ \
BEGIN RETURN QUERY SELECT "Game".game_name FROM "Game" INNER JOIN "Category" on "Game".category_id="Category".category_id \
WHERE "Category".category_name = cat_name;\
END;$$ \
LANGUAGE 'plpgsql';'''

print(mapper_stmt)
mapper_results = engine.execute(mapper_stmt)
```

```
CREATE OR REPLACE FUNCTION get_games_for_cat(cat_name VARCHAR)RETURNS TABLE (game_name VARCHAR) AS $$ BEGIN RETURN QUERY SELECT "Game".ga
me_name FROM "Game" INNER JOIN "Category" on "Game".category_id="Category".category_id WHERE "Category".category_name = cat_name;END;$$ L
ANGUAGE 'plpgsql';
```

```
mapper_stmt = "Select * from get_games_for_cat('Shooter') limit 30"
print(mapper_stmt)
mapper_results = engine.execute(mapper_stmt).fetchall()
print(mapper_results)
```

```
Select * from get_games_for_cat('Shooter') limit 30
[('Duck Hunt',), ('Call of Duty: Modern Warfare 3',), ('Call of Duty: Black Ops',), ('Call of Duty: Black Ops 3',), ('Call of Duty: Black
Ops II',), ('Call of Duty: Black Ops II',), ('Call of Duty: Modern Warfare 2',), ('Call of Duty: Modern Warfare 3',), ('Call of Duty: Bla
ck Ops',), ('Halo 3',), ('Call of Duty: Modern Warfare 2',), ('Call of Duty: Ghosts',), ('Halo: Reach',), ('Halo 4',), ('Call of Duty: Gh
osts',), ('Call of Duty 4: Modern Warfare',), ('Halo 2',), ('GoldenEye 007',), ('Star Wars Battlefront (2015)',), ('Call of Duty: Advance
d Warfare',), ('Call of Duty: World at War',), ('Battlefield 3',), ('Call of Duty: Black Ops 3',), ('Battlefield 3',), ('Medal of Honor:
Frontline',), ('Gears of War 2',), ('Call of Duty 4: Modern Warfare',), ('Halo: Combat Evolved',), ('Halo 3: ODST',), ('Gears of War
3',)]
```

```
mapper_stmt = "Select * from get_games_for_cat('Role-Playing') limit 30"
print(mapper_stmt)
mapper_results = engine.execute(mapper_stmt).fetchall()
print(mapper_results)
```

```
Select * from get_games_for_cat('Role-Playing') limit 30
[('Pokemon Red/Pokemon Blue',), ('Pokemon Gold/Pokemon Silver',), ('Pokemon Diamond/Pokemon Pearl',), ('Pokemon Ruby/Pokemon Sapphire',),
('Pokemon Black/Pokemon White',), ('Pokémon Yellow: Special Pikachu Edition',), ('Pokemon X/Pokemon Y',), ('Pokemon Omega Ruby/Pokemon Al
pha Sapphire',), ('Pokemon FireRed/Pokemon LeafGreen',), ('Final Fantasy VII',), ('The Elder Scrolls V: Skyrim',), ('Pokemon Black 2/Poke
mon White 2',), ('Final Fantasy X',), ('Final Fantasy VIII',), ('Pokémon Platinum Version',), ('Fallout 4',), ('The Elder Scrolls V: Skyr
im',), ('Pokémon Emerald Version',), ('Kingdom Hearts',), ('Pokémon Crystal Version',), ('World of Warcraft',), ('Final Fantasy XII',),
('Dragon Quest IX: Sentinels of the Starry Skies',), ('Monster Hunter Freedom Unite',), ('Final Fantasy XIII',), ('Final Fantasy IX',),
('Final Fantasy X-2',), ('Dragon Quest VIII: Journey of the Cursed King',), ('Diablo III',), ('Fable III',)]
```