Generowanie bazy danych

Czerwiec 2023

1 Opis działalności sklepu

W generowanej przez nas bazie znajdować się będą dane dotyczące sklepu „Geeks & Dragons", oferującego gry planszowe. Działalność firmy skupia się na sprzedaży i wypożyczeniach gier oraz organizacji turniejów. Przyjmujemy, że firma posiada jedną placówkę ulokowaną we Wrocławiu, która powstała 29.04.2022r.

2 Tworzenie tabel

Generowanie bazy danych rozpoczęłyśmy od utworzenia pustych tabel przy pomocy odpowiednich komend w języku SQL. W tym celu każdej kolumnie przyporządkowałyśmy odpowiedni typ danych, jakie będzie ona przechowywała oraz ustaliłyśmy, czy dane te są unikalne i czy mogą występować wśród nich braki. Na tej podstawie napisałyśmy kod znajdujący się w pliku create_tables.sql. Poniżej prezentujemy przykład kodu dla jednej z tabel.

```
CREATE OR REPLACE TABLE team03.customer
(
customer_id INT UNSIGNED NOT NULL PRIMARY KEY UNIQUE AUTO_INCREMENT,
first_name VARCHAR(255) NOT NULL,
last_name VARCHAR(255) NOT NULL,
email VARCHAR(255) UNIQUE NOT NULL,
phone_number CHAR(13) UNIQUE NOT NULL,
address_id INT UNSIGNED NOT NULL
)ENGINE = InnoDB;
```

Rysunek 1: Kod odpowiedzialny za tworzenie tabeli *Customer*.

3 Generowanie danych

W celu wypełnienia naszych tabel danymi zdecydowałyśmy się napisać skrypt w języku Python. Całość kodu można znaleźć w pliku generate_data.py. W dalszej części tego rozdziału prezentujemy funkcje odpowiedzialne za tworzenie poszczególnych tabel. Część z nich korzysta z funkcji zaimplementowanych w dodatkowych bibliotekach lub z danych

statystycznych zapisanych w zewnętrznych plikach, dlatego poniżej prezentujemy kod potrzebny do prawidłowego działania tych funkcji.

```
import pandas as pd
import numpy as np
import random
from urllib import parse
from sqlalchemy import create_engine
from datetime import timedelta, datetime
from unidecode import unidecode

games = pd.read_csv("data/games_to_choose.csv")
male_names = pd.read_csv("data/male_names.csv")
female_names = pd.read_csv("data/female_names.csv")
male_surnames = pd.read_csv("data/male_surnames.csv")
female_surnames = pd.read_csv("data/female_surnames.csv")
addresses = pd.read_csv("data/addresses.csv")
```

Rysunek 2: Potrzebne biblioteki i pliki z danymi.

3.1 Address

Tabela *Address* przechowuje dane dotyczące miejsc zamieszkania naszych klientów i pracowników. Składa się ona z następujących kolumn:

- **Address_ID** - identyfikator adresu,
- **Street** - nazwa ulicy,
- **Street_number** - numer budynku,
- **City** - miasto,
- **Postal_code** - kod pocztowy.

Do wygenerowania adresów wykorzystałyśmy dane udostępnione na stronie Urzędu Miejskiego Wrocławia. Z dostępnych tam 71488 adresów losowo wybrałyśmy 503. Każdy z adresów miał takie samo prawdopodobieństwo bycia wylosowanym i żaden z nich nie mógł być wylosowany kilkukrotnie.

Za tę część generowania danych odpowiedzialna była poniższa funkcja.

```
def generate_addresses(n):
    idx = random.sample(range(len(addresses)), n)
    df = addresses.iloc[idx, [2, 3, 4]]
    df = df.rename(columns = {"ULICA_NAZWA": "Street", "NUMER_ADR": "Street_number", "KOD_POCZTOWY": "Postal_code"})
    df.insert(0, "Address_ID", range(1, n+1))
    df.insert(3, "City", "Wrocław")
    df = df.set_index("Address_ID")
    return df
```

Rysunek 3: Kod odpowiedzialny za generowanie tabeli *Address*.

3.2 Customer

Tabela *Customer* przechowuje dane osobowe i kontaktowe naszych zarejestrowanych klientów. Przyjmujemy, że jest ich 500. Tabela składa się z następujących kolumn:

- **Customer_ID** - identyfikator klienta,
- **First_name** - imię klienta,
- **Last_name** - nazwisko klienta,
- **Email** - adres poczty elektronicznej klienta,
- **Phone_number** - numer telefonu klienta,
- **Address_ID** - identyfikator adresu zamieszkania klienta.

Do stworzenia tej tabeli wykorzystałyśmy dane dotyczące imion i nazwisk występujących w rejestrze PESEL. Każde z nich miało prawdopodobieństwo bycia wylosowanym proporcjonalne do liczby wystąpień tego imienia lub nazwiska. Liczba kobiet będących naszymi klientami to część całkowita zmiennej losowej z rozkładu normalnego o średniej $0.5n$ i odchyleniu standardowym $0.02n$, gdzie n stanowi liczbę wszystkich klientów.

Adres poczty elektronicznej klienta zaczyna się z równym prawdopodobieństwem od jego imienia lub tylko pierwszej litery jego imienia, następnie występuje kropka i nazwisko klienta (wszystko bez polskich znaków). Średnio w połowie przypadków na końcu dodana jest jeszcze liczba wylosowana z rozkładu geometrycznego z parametrem 0.5. Nazwy domenowe serwerów poczty elektronicznej naszych klientów to z równymi prawdopodobieńswami wp.pl, gmail.com oraz onet.pl.

Każdy numer telefonu składa się z numeru kierunkowego +48 i 9 losowo wybranych liczb, przy czym pierwsza z nich jest wybrana z zakresu od 5 do 8, a pozostałe z zakresu od 0 do 9.

Identyfikator adresu jest losowo wybranym indeksem z tabeli *Address*.

Za tę część generowania danych odpowiedzialny był poniższy kod.

```python
def rand_email(name, surname):
    return (unidecode(random.choice([name, name[0]])) + "." + unidecode(surname) +
            random.choice(["", str(np.random.geometric(0.5))]) +
            random.choice(["@wp.pl", "@gmail.com", "@onet.pl"])).lower()

def rand_females(n):
    names = random.choices(female_names["IMIĘ PIERWSZE"], weights = female_names["LICZBA WYSTĄPIEŃ"], k = n)
    surnames = random.choices(female_surnames["Nazwisko aktualne"], weights = female_surnames["Liczba"], k = n)
    emails = [rand_email(name, surname) for name, surname in zip(names, surnames)]
    return [name + "*" + surname + "*" + email for name, surname, email in zip(names, surnames, emails)]

def rand_males(n):
    names = random.choices(male_names["IMIĘ PIERWSZE"], weights = male_names["LICZBA WYSTĄPIEŃ"], k = n)
    surnames = random.choices(male_surnames["Nazwisko aktualne"], weights = male_surnames["Liczba"], k = n)
    emails = [rand_email(name, surname) for name, surname in zip(names, surnames)]
    return [name + "*" + surname + "*" + email for name, surname, email in zip(names, surnames, emails)]

def rand_people(n):
    female_number = int(np.random.normal(0.5 * n, 0.02 * n))
    people = rand_females(female_number) + rand_males(n - female_number)
    random.shuffle(people)
    return people

def rand_phone_numbers(n):
    return ["+48" + random.choice(["5", "6", "7", "8"]) + str(random.randint(10**7, 10**8-1)) for I in range(n)]

def generate_customers(n, addresses_number):
    df = pd.DataFrame()
    df["Customer_ID"] = range(1, n+1)
    df[["First_name", "Last_name", "Email"]] = pd.Series(rand_people(n)).str.split("*", expand = True)
    df["Phone_number"] = rand_phone_numbers(n)
    df["Address_ID"] = random.sample(range(1, addresses_number + 1), n)
    df = df.set_index("Customer_ID")
    return df
```

Rysunek 4: Kod odpowiedzialny za generowanie tabeli *Customer*.

3.3 Employee

Tabela *Employee* przechowuje dane osobowe i kontaktowe naszych pracowników. Przyjmujemy, że jest ich 3. Tabela składa się z następujących kolumn:

- **Employee_ID** - identyfikator pracownika,
- **First_name** - imię pracownika,
- **Last_name** - nazwisko pracownika,
- **Email** - adres poczty elektronicznej pracownika,
- **Phone_number** - numer telefonu pracownika,
- **Address_ID** - identyfikator adresu zamieszkania pracownika,
- **Employment_date** - data zatrudnienia pracownika,
- **Dismissal_date** - data zwolnienia pracownika lub NULL, jeśli pracownik jest nadal zatrudniony,
- **Salary** - zarobki brutto pracownika lub NULL, jeśli pracownik został zwolniony.

Pierwsze 6 kolumn tej tabeli zostało wygenerowane analogicznie do tabeli *Customer*. Wszyscy pracownicy zostali zatrudnieni w pierwszym dniu działalności sklepu i żaden z nich nie został zwolniony. Nasi pracownicy otrzymują wynagrodzenie w wysokości 4500zł przez pierwsze pół roku pracy, a po tym czasie 5500zł.

Za tę część generowania danych odpowiedzialny był poniższy kod.

```
def rand_dates_and_salaries(n, first_date):
    result = []
    for i in range(n):
        empl_date = first_date
        dism_date = None
        salary = None if dism_date else (5500 if datetime.now() - empl_date > timedelta(weeks = 26) else 4500)
        result.append((empl_date, dism_date, salary))
    return result

def generate_employees(n, addresses_number, first_date):
    df = pd.DataFrame()
    df["Employee_ID"] = range(1, n+1)
    df[["First_name", "Last_name", "Email"]] = pd.Series(rand_people(n)).str.split("*", expand = True)
    df["Phone_number"] = rand_phone_numbers(n)
    df["Address_ID"] = random.sample(range(1, addresses_number + 1), n)
    df[["Employment_date", "Dismissal_date", "Salary"]] = rand_dates_and_salaries(n, first_date)
    df = df.set_index("Employee_ID")
    return df
```

Rysunek 5: Kod odpowiedzialny za generowanie tabeli *Employee*.

3.4 Game

Tabela *Game* zawiera dane o wszystkich grach, które nasz sklep może sprzedawać, wypożyczać lub wykorzystywać na turniejach.

- **Game_ID** - identyfikator gry,
- **Game_title** - tytuł gry,
- **Min_players** - minimalna liczba graczy,
- **Max_players** - maksymalna liczba graczy,
- **Playing_time** - szacowany czas trwania rozgrywki,
- **Year_published** - rok wydania,
- **Item_type** - typ gry, tzn. gra podstawowa (wartość "standalone") lub rozszerzenie (wartość "extension"),
- **Publisher** - wydawca gry,
- **Language** - język wydania,
- **Rental_price** - cena wypożyczenia,
- **Price** - cena zakupu,

- **Tournament_game** - znacznik określający, czy nasz sklep może organizować turnieje z daną grą.

Tabela *Game* została wygenerowana na podstawie danych znalezionych na stronie https://boardgamegeek.com. Wybrałyśmy interesujące nas kolumny i 5 tytułów, które będą rozgrywane podczas turniejów. Następnie wylosowałyśmy 45 tytułów, które nasz sklep będzie mógł sprzedawać i wypożyczać.

```
def choose_games(games_to_choose, num_of_games):
    tournament_games = games_to_choose[games_to_choose["Tournament_game"] == 1]
    not_tournament_games = games_to_choose[games_to_choose["Tournament_game"] == 0]
    games_sample = not_tournament_games.sample(num_of_games - 5)
    not_tournament_id = list(games_sample["Game_ID"])
    tournament_id = list(tournament_games["Game_ID"])
    all_games = not_tournament_id + tournament_id
    game_df = games[games["Game_ID"].isin(all_games)].iloc[:, [0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]]
    game_df["Game_ID"] = np.arange(1, len(game_df) + 1)
    game_df.set_index("Game_ID", inplace = True)
    all_games = list(game_df[game_df["Tournament_game"] == 0].index) + list(game_df[game_df["Tournament_game"] == 1].index)
    return all_games, game_df
```

Rysunek 6: Kod odpowiedzialny za generowanie tabeli *Game*.

3.5 Product

Tabela *Product* przechowuje dane dotyczące wszystkich produktów, które znajdują się w asortymencie sklepu lub kiedyś się w nim znajdowały.

- **Product_ID** - identyfikator produktu,
- **For_rent** - znacznik określający, czy produkt jest przeznaczony do wypożyczenia,
- **For_sale** - znacznik określający, czy produkt jest przeznaczony do sprzedaży,
- **For_tournament** - znacznik określający, czy produkt jest przeznaczony na turnieje,
- **Supply_date** - data zaopatrzenia,
- **Game_ID** - identyfikator gry,
- **Purchase_ID** - identyfikator zakupu; jeśli produkt nie został zakupiony, ma wartość NULL.

Najpierw stworzyłyśmy tabelę „Product” z początkowym wyposażeniem sklepu. Zaczęłyśmy z 500 produktami na stanie, z czego 25 sztuk gier było przeznaczonych na turnieje – po 5 każdego tytułu turniejowego. Pozostałe 475 gier zostało wylosowanych z tabeli *Game* i każda z równym prawdopodobieństwem jest klasyfikowana jako do wypożyczenia lub do kupienia.

```python
def choose_products(games_to_choose_from, num_of_products, first_date):
    not_tournament_id = games_to_choose_from[-5:]
    tournament_products = [not_tournament_id[i] for i in np.arange(5)]*5
    for_rent_tournament = np.zeros(25)
    for_sale_tournament = np.zeros(25)
    for_tournament_tournament = [1 for i in np.arange(25)]

    not_tournament_products = random.choices(games_to_choose_from, k=num_of_products - 25)
    for_tournament_nt = [0 for i in np.arange(num_of_products - 25)]
    supply_date= [first_date for i in np.arange(num_of_products)]

    for_sale_nt = np.zeros(num_of_products - 25)
    for_rent_nt = np.zeros(num_of_products - 25)

    for i in np.arange(num_of_products - 25):
        rand = random.randint(0, 1)
        for_sale_nt[i] = rand
        if for_sale_nt[i] == 0:
            for_rent_nt[i] = 1
        else:
            for_rent_nt[i] = 0

    for_rent = list(for_rent_tournament) + list(for_rent_nt)
    for_sale = list(for_sale_tournament) + list(for_sale_nt)
    for_tournament = for_tournament_tournament + for_tournament_nt
    all_games_products = tournament_products + not_tournament_products

    product_df = pd.DataFrame()
    product_df["Product_ID"] = np.arange(1, num_of_products + 1)
    product_df["For_rent"] = for_rent
    product_df["For_sale"] = for_sale
    product_df["For_tournament"] = for_tournament
    product_df["Supply_date"] = supply_date
    product_df["Game_ID"] = all_games_products
    product_df["Purchase_ID"] = np.nan

    product_df["Supply_date"] = pd.to_datetime(product_df["Supply_date"])
    product_df["For_rent"] = pd.to_numeric(product_df["For_rent"], downcast='integer')
    product_df["For_sale"] = pd.to_numeric(product_df["For_sale"], downcast='integer')

    return product_df
```

Rysunek 7: Kod odpowiedzialny za generowanie początkowej tabeli *Product*.

Dodatkowo każdego dnia (oprócz niedzieli) do sklepu przychodziła dostawa nowych gier przeznaczonych do sprzedaży. Liczba nowych produktów została wylosowana z rozkładu Poissona o średniej 10, natomiast ich tytuły z tabeli *Game*. Kod zasilający tabelę znajduję się na rysunkach 8, 9 i 10.

3.6 Rental

Tabela *Rental* przechowuje dane dotyczące wypożyczeń.

- **Rental_ID** - identyfikator wypożyczenia,
- **Rental_date** - data wypożyczenia,
- **Return_date_expected** - oczekiwana data zwrócenia wypożyczenia,
- **Return_dane_actual** - data zwrócenia wypożyczenia; jeśli produkt nie został jeszcze zwrócony, przyjmuje wartość NULL,

- **Customer_ID** - identyfikator klienta, który wypożyczył produkt/produkty,
- **Employee_ID** - identyfikator pracownika, który obsłużył wypożyczenie.

Dla każdego dnia działania sklepu, oprócz niedziel, losowałyśmy dane dotyczące wypożyczeń. Liczba dokonanych wypożyczeń była równa realizacji zmiennej losowych z rozkładu Poissona o średniej 10. Każde wypożyczenie miało losowa przypisanego klienta z tabeli *Customer* i odpowiedzialnego pracownika z tabeli *Employee* oraz obejmowało liczbę gier losowaną z rozkładu geometrycznego o średniej 1.3. Nasz sklep oferuje wypożyczenia na 3 dni. Każde wypożyczenie zostało zwrócone z prawdopodobieństwem 0.5 po 3 dniach i z prawdopodobieństwem 0.5 po liczbie dni równych realizacji zmiennej losowej z rozkładu geometrycznego o średniej 3.

Tabela jest generowana za pomocą funkcji z rysunków 8, 9 i 10.

3.7 Rental_Product_Rel

Między tabelami *Rental* i *Product* zachodzi relacja wiele do wielu. Wymusiło to stworzenie tabeli relacyjnej *Rental_Product_Rel*.

- **Relation_ID** - identyfikator relacji,
- **Product_ID** - identyfikator produktu,
- **Rental_ID** - identyfikator wypożyczenia.

Tabelę generuje kod z rysunków 8, 9 i 10.

3.8 Purchase

Tabela *Purchase* przechowuje dane dotyczące zakupów dokonywanych w naszym sklepie.

- **Purchase_ID** - identyfikator zakupu,
- **Customer_ID** - identyfikator pracownika,
- **Purchase_date** - data zakupu,
- **Employee_ID** - identyfikator klienta; przyjmuje wartość NULL, jeśli nie znamy danych klienta.

Dane do tabeli *Purchase* zostały wygenerowane analogicznie do tabeli *Rental*. Zmiana zachodzi jedynie dla przypisywania klienta do zakupu – z prawdopodobieństwem 0.7 nie jest on zarejestrowany w naszej bazie (nie musi, skoro nic nie wypożycza ani nie bierze udziału w turniejach) lub nie chce podać swoich danych i przypisywana jest mu wartość NULL.

Tabelę *Purchase* jest generowana za pomocą funkcji z rysunków 8, 9 i 10.

```python
def product_df_time(product_df, first_date):

    cust_num = 501 #real customer number is cust_num - 1
    worker_num = 4 #real employee number is worker_num - 1

    rental_id = [0]
    rental_date = []
    return_date_expected = []
    return_date_actual = []
    customer_rent_id = []
    worker_rent_id = []

    relation_id = [0]
    product_id_rel = []
    rental_id_rel = []

    product_and_return = {}

    purchase_id = [0]
    purchase_date = []
    customer_purch_id = []
    worker_purch_id = []

    num_to_rent = np.random.poisson(10, 365)
    num_to_sale = np.random.poisson(10, 365)

    num_to_supply = np.random.poisson(10, 365)

    j = 0
    for days_count in np.arange(1, 366):

        if days_count % 7 != 2:

            new_date = first_date + timedelta(days = int(days_count))

            games_rentable = product_df[product_df["For_rent"] == 1]

            num_of_games_to_rent = np.random.geometric(1/1.3, num_to_rent[j]) #ile gier na wypożyczenie?

            while sum(num_of_games_to_rent) > len(games_rentable):
                num_of_games_to_rent = num_of_games_to_rent[:-1]

            random_return = np.random.random(len(num_of_games_to_rent))
            rent_cust_id = np.random.randint(1, cust_num, len(num_of_games_to_rent))
            rent_worker_id = np.random.randint(1, worker_num, len(num_of_games_to_rent))

            games_to_rent = games_rentable["Product_ID"].sample(sum(num_of_games_to_rent)) #jakie gry wypożyczono?

            k = 0
            i = 0
            for rent in num_of_games_to_rent:

                games_to_this_rent = np.array(games_to_rent)[k:k+rent]
```

Rysunek 8: Kod odpowiedzialny za zasilanie tabeli *Product* i generowanie tabel *Rental*, *Rental_Product_Rel*, *Purchase*. [1/3]

```
        available_products = []

        for game in games_to_this_rent: #sprawdzamy, czy produkty są dostępne
            if game not in list(product_and_return.keys()) or product_and_return[game] < new_date:
                available_products.append(game)

        num_of_products = len(available_products)


        if num_of_products > 0:

            rental_id.append(rental_id[-1]+1)
            rental_date.append(new_date)
            return_date_expected.append(new_date + timedelta(days = 3))
            customer_rent_id.append(rent_cust_id[i])
            worker_rent_id.append(rent_worker_id[i])

            if random_return[i] < 0.5:
                return_date_actual.append(return_date_expected[-1])
                product_and_return[game] = return_date_expected[-1]

            else:
                return_date = new_date + timedelta(days = np.random.geometric(1/3))
                return_date_actual.append(return_date)
                product_and_return[game] = return_date

            relation_id = relation_id + [relation_id[-1] + i for i in np.arange(1, num_of_products+1)]
            rental_id_rel = rental_id_rel + [rental_id[-1]]*num_of_products
            product_id_rel = product_id_rel + available_products

        k += rent
        i += 1


    #zakup gier

    games_salable = product_df[(product_df["For_sale"] == 1) & (pd.isna(product_df["Purchase_ID"]))]
    num_of_games_to_sale = np.random.geometric(1/1.3, num_to_sale[j]) #ile gier na wypożyczenie?
    purch_cust_id = np.random.randint(1, cust_num, len(num_of_games_to_sale))
    purch_worker_id = np.random.randint(1, worker_num, len(num_of_games_to_sale))
    random_cust = np.random.random(len(num_of_games_to_sale))

    while sum(num_of_games_to_sale) > len(games_salable):
        num_of_games_to_sale = num_of_games_to_sale[:-1]

    games_to_sale = games_salable["Product_ID"].sample(sum(num_of_games_to_sale)) #jakie gry wypożyczono?

    k = 0
    i = 0

    for purchase in num_of_games_to_sale:

        games_to_this_purchase = np.array(games_to_sale)[k:k+purchase]
```

Rysunek 9: Kod odpowiedzialny za zasilanie tabeli *Product* i generowanie tabel *Rental*, *Rental_Product_Rel*, *Purchase* [2/3].

```
            purchase_id.append(purchase_id[-1]+1)
            purchase_date.append(new_date)

            if random_cust[i] < 0.3:
                customer_purch_id.append(purch_cust_id[i])
            else:
                customer_purch_id.append(np.nan)

            worker_purch_id.append(purch_worker_id[i])


            for game_product in games_to_this_purchase:
                product_df.iloc[game_product-1, product_df.columns.get_loc('Purchase_ID')] = purchase_id[-1]

            k += purchase
            i += 1

        games_to_supply = random.choices(all_games, k = num_to_supply[j])

        max_product_id = max(np.array(product_df["Product_ID"]))
        new_df = pd.DataFrame()
        new_df["Product_ID"] = np.arange(max_product_id + 1, max_product_id + num_to_supply[j] + 1)
        new_df["For_sale"] = [1 for _ in np.arange(num_to_supply[j])]
        new_df["For_rent"] = [0 for _ in np.arange(num_to_supply[j])]
        new_df["For_tournament"] = [0 for _ in np.arange(num_to_supply[j])]
        new_df["Supply_date"] = [new_date for _ in np.arange(num_to_supply[j])]
        new_df["Game_ID"] = games_to_supply
        new_df["Purchase_ID"] = np.nan

        new_df["Supply_date"] = pd.to_datetime(new_df["Supply_date"])
        new_df["For_rent"] = pd.to_numeric(new_df["For_rent"], downcast='integer')
        new_df["For_sale"] = pd.to_numeric(new_df["For_sale"], downcast='integer')

        product_df = pd.concat([product_df, new_df], ignore_index=True)

    j += 1

rental_df = pd.DataFrame({"Rental_ID":rental_id[1:], "Rental_date":rental_date,
                          "Return_date_expected":return_date_expected,
                          "Return_date_actual":return_date_actual, "Customer_ID":customer_rent_id,
                          "Employee_ID":worker_rent_id}).set_index("Rental_ID")
rental_product_rel_df = pd.DataFrame({"Relation_ID":relation_id[1:], "Product_ID":product_id_rel,
                                      "Rental_ID":rental_id_rel}).set_index("Relation_ID")
purchase_df = pd.DataFrame({"Purchase_ID":purchase_id[1:], "Purchase_date":purchase_date,
                            "Customer_ID":customer_purch_id,
                            "Employee_ID":worker_purch_id}).set_index("Purchase_ID")
product_df["Product_ID"] = range(1, len(product_df.index) + 1)
product_df = product_df.dropna(subset = ["For_rent", "For_sale", "For_tournament"]).set_index("Product_ID")

return rental_df, rental_product_rel_df, purchase_df, product_df
```

Rysunek 10: Kod odpowiedzialny za zasilanie tabeli *Product* i generowanie tabel *Rental*, *Rental_Product_Rel*, *Purchase* [3/3].

3.9 Tournament i Score

Tabela *Tournament* zawiera dane dotyczące turniejów zorganizowanych przez nasz sklep oraz tych, które jeszcze się nie odbyły, ale są już zaplanowane.

- **Tournament_ID** - identyfikator turnieju,
- **Tournament_date** - data turnieju,
- **Ticket_price** - cena za bilet wstępu,
- **Tournament_cost** - koszt organizacji turnieju,
- **Game_ID** - identyfikator gry, która jest rozgrywana podczas turnieju.

Tabela *Score* przechowuje dane o wynikach turniejów.

- **Score_ID** - identyfikator wyniku,
- **Tournament_ID** - identyfikator turnieju,
- **Customer_ID** - identyfikator klienta,
- **Score** - otrzymany wynik.

W każdy piątek i w każdą sobotę z prawdopodobieństwem 0.3 nasz sklep zorganizował turniej, na którym była rozgrywana losowo wybrana gra turniejowa (z pięciu dostępnych tytułów). W każdym z nich brała udział liczba uczestników równa realizacji zmiennej losowej z rozkładu Poissona o średniej 25, ale nie mniejsza niż 10. Każdy uczestnik był zarejestrowany w bazie klientów i po zakończonych rozgrywkach został mu przypisywany wynik, którego wysokość zależy od miejsca, które zajął w turnieju. Ponadto turnieje zostały zaplanowane na kolejne cztery miesiące.

```
def create_tournament_and_score_df(first_date):
    tournament_id = [0]
    tournament_date = []
    ticket_price = []
    game_id = []
    tournament_cost = []

    cust_num = 501 #real customer number is cust_num - 1

    tournament_id_score = []
    customer_id_score = []
    score = []

    for days_count in np.arange(1, 500):
        if days_count % 7 == 1 or days_count % 7 == 0:
            new_date = first_date + timedelta(days = int(days_count))

            if np.random.random() < 0.3:
                tournament_id.append(tournament_id[-1] + 1)
                tournament_date.append(new_date)
                ticket_price.append(np.random.poisson(18))
                game_id.append(np.random.choice(products_df[products_df["For_tournament"] == 1].loc[:, "Game_ID"].unique()))
                tournament_cost.append(ticket_price[-1]*np.random.poisson(10))

                if days_count < 366:
                    num_of_players = np.max([10, np.random.poisson(25)])

                    while num_of_players >= cust_num - 1:
                        num_of_players =- 1

                    if num_of_players > 0:
                        customer_id_score = customer_id_score + list(np.random.choice(np.arange(cust_num), num_of_players, replace=False))
                        score = score + list(np.arange(num_of_players))
                        tournament_id_score = tournament_id_score + [tournament_id[-1] for _ in np.arange(num_of_players)]

    score_id = np.arange(1, len(score) + 1)
    tournament_df = pd.DataFrame({"Tournament_ID":tournament_id[1:], "Tournament_date":tournament_date,
                                  "Ticket_price":ticket_price, "Tournament_cost":tournament_cost, "Game_ID":game_id}).set_index("Tournament_ID")
    score_df = pd.DataFrame({"Score_ID":score_id, "Tournament_ID":tournament_id_score,
                             "Customer_ID":customer_id_score, "Score":score}).set_index("Score_ID")

    return tournament_df, score_df
```

Rysunek 11: Kod odpowiedzialny za generowanie tabel *Tournament* i *Score*.

4 Integracja Pythona z bazą danych

Ostatnim krokiem w procesie generowania bazy danych było połączenie się z nią z poziomu naszego skryptu i uzupełnienie pustych tabel danymi, które wygenerowałyśmy.

W tym celu postanowiłyśmy wykorzystać bibliotekę sqlalchemy, a niżej przedstawiony kod pozwolił nam połączyć się z bazą danych.

```
engine = create_engine("mysql+pymysql://{user}:{password}@{account}:3306/team03".format(
    user = "team03",
    password = parse.quote("te@m0e"),
    account = "giniewicz.it"
    )
)

conn = engine.connect()
```

Rysunek 12: Kod potrzebny do połączenia się z bazą danych.

Następnie wystarczyło dla każdej tabeli napisać kod, który usuwa z niej aktualne wartości (aby można było generować nowe dane wielokrotnie) i przypisuje nowe. Przykład takiego kodu dla jednej z tabel możemy zobaczyć poniżej.

```
customer_df = generate_customers(500, 503)

conn.execute("TRUNCATE TABLE customer")
customer_df.to_sql("customer", engine, if_exists = "append")
```

Rysunek 13: Kod uzupełniający tabelę *Customer* wygenerowanymi danymi.