W celu ułatwienia uzupełniania tabel zdefiniowaliśmy w Jupiter Notebook'u funkcje:

- "create_db_connection" - odpowiedzialne za stworzenie połączenia z bazą danych,

```
def create_db_connection(host_name, user_name, user_password, db_name):
    connection = None
    try:
        connection = mysql.connector.connect(
            host=host_name,
            user=user_name,
            passwd=user_password,
            database=db_name
        )
        print("MySQL Database connection successful")
    except Error as err:
        print(f"Error: '{err}'")

    return connection
```

- "read_query" - odczytuje wyniki zapytania do bazy danych (podanego jako string) ,

```
def read_query(connection, query):
    cursor = connection.cursor()
    result = None
    try:
        cursor.execute(query)
        result = cursor.fetchall()
        return result
    except Error as err:
        print(f"Error: '{err}'")
```

- "execute_query" - wykonuje polecenie podane jako string,

```
def execute_query(connect, query):
    cursor = connect.cursor()
    try:
        cursor.execute(query)
        connect.commit()
        print("Query successful")
    except Error as err:
        print(f"Error: '{err}'")
```

- "upgrade" - stosowane do modyfikowania już wprowadzonych danych.

```
def upgrade(kolumny_zmiany,tabela,warunek):
    komenda="UPDATE "+str(tabela)+""" SET
    """
    for zmiana in kolumny_zmiany:
        komenda=komenda+str(zmiana)+""" , """
    komenda=komenda[:-2]+"WHERE "+str(warunek)+" ;"
    projekt=create_db_connection("giniewicz.it", "team06","te@m0g" , "team06")
    execute_query(projekt,komenda)
```

Podczas generowania tabeli "Customers" losowaliśmy płeć klienta przy pomocy zmiennej z rozkładu jednorodnego na odcinku [0,1) (U[0,1]) i porównywaliśmy jej wartość z 1/2 , a następnie losowaliśmy imię i nazwisko mężczyzny (lub kobiety) z odpowiednio 'first_names_M' [2] i 'last_names_M' [1] (lub 'first_names_K' [3] i 'last_names_K'[1]). 10 najczęściej występujących imion i nazwisk w tych listach posiada dwukrotnie większe prawdopodobieństwo wystąpienia niż pozostałe. Wyjątkiem jest argument 'first_names_K', które przez swoje uszeregowanie pod względem alfabetycznym, nie daje możliwości odczytania z niego 10 najpopularniejszych imion kobiecych. Założyliśmy , że w okresie 21.04.2020 - 14.02.2022 w naszym sklepie 1850 klientów dokonało przynajmniej jednej transakcji.

Ze względu na niewielką liczbę pracowników(8) nie było potrzeby generowania wielu zmiennych, więc tabela "Employees" została uzupełniona ręcznie.

Do stworzenia tabeli "Shop_inventory"(oprócz szachów), wprowadziliśmy również listę zawierającą 80 najpopularniejszych gier planszowych. Każdej grze przypisywana jest cena sprzedaży, losowana jest ona spośród czterech różnych opcji oraz powiązana z nią jest miesięczna cena wypożyczenia gry. Natomiast aktualna ilość egzemplarzy, znajdująca się w magazynie, generowaliśmy dzięki zmiennej:

$$\left\lfloor \frac{100 - U(0, Numer\ Gry\ w\ Rankingu)}{5} \right\rfloor$$

.

Podczas tworzenia tabeli "Game_purchase" dla każdego dnia losowaliśmy liczbę klientów przypadających na dany dzień z rozkładu Poissona z rosnącym w tempie geometrycznym parametrem

$$\lambda = 2 \times 3^{\frac{Numer\ Dnia+1}{300}}$$

.

Przy pomocy działania operacji mod 7 sprawdzaliśmy , czy dniem tygodnia jest niedziela (ponieważ założyliśmy , iż nasz sklep nie funkcjonuje w niedziele) , która skutkowała iteracyjnym przejściem do następnego dnia. Natomiast "Customer_id" klientów generowaliśmy przy pomocy funkcji:

$$min(1 + \lfloor U(0, 2 \times Numer\ Dnia + 518) \rfloor, Ilość\ Dotychczasowych\ Klijentów + 1)$$

.

Aby zapewnić , iż wszyscy klienci byli nimi naprawdę, pod koniec każdego dnia w pętli sprawdzamy czy liczba pozostałych klientów nie przekracza trzykrotności liczby pozostałych dni i ewentualnie dodajemy(do 3) transakcji tego dnia(działo się tak niemal wyłącznie w ostatnie dni).

Numer gry losowaliśmy z 81 możliwości , dając pierwszym 51 pozycjom dwukrotnie większą wagę niż pozostałym 30. Następnie przy pomocy funkcji "read_query" odczytywaliśmy cenę przypisaną wygenerowanemu 'Game_id'. Ostatecznie losowaliśmy pracownika uczestniczącego w transakcji dzięki zmiennej :

$$\left\lfloor U\left(0, \left\lfloor 3 + \frac{192 + Numer\ Zakupu}{1000} \right\rfloor\right) \right\rfloor + 1$$

.

Definiując ją mieliśmy na celu odwzorowanie stopniowego zatrudniania kolejnych pracowników wraz z biegiem czasu. Następnie dzięki wszystkim danym tworzyliśmy string, będący parametrem funkcji "execute_query", dzięki której uzupełnialiśmy tabelę.

Proces generowania tabeli "Game_rental" przebiegał podobnie. Nie licząc, wstawienia czterokrotnie większej wartości parametru w rozkładzie Poissona generującym liczbę klientów i zmodyfikowaniem rozkładu, dzięki któremu generowaliśmy "Employee_id" na :

$$\left\lfloor U\left(0, \left\lfloor 3 + \frac{402 + Numer\ Zakupu}{3300} \right\rfloor\right) \right\rfloor + 1$$

.

Oprócz tego konieczne było również wylosowanie długości na jaką będzie wypożyczana gra przy pomocy wzoru:

$$min(1 + \lfloor\ |N(0,1)|\ \rfloor, 3) \times 30 - \lfloor U(0,7) \rfloor$$

.

Wartość tę następnie wykorzystujemy do wyznaczenia dnia zwrotu który jeśli wypada w niedziele jest przekładany na następny dzień. Natomiast jeśli dzień zwrotu wypadłby po 14.02.2022 wstawiona zostanie wartość "NULL". Cenę wypożyczenia odczytujemy dzięki funkcji "read_query" w sposób identyczny jak w tabeli "Game_purchase" oraz przemnażamy tę wartość przez ilość miesięcy, na które gra została wypożyczona.

Tabele "Game_tournaments" uzupełnialiśmy ręcznie w taki sposób by turnieje odbywały się od czerwca 2020 roku w każdą pierwszą sobotę miesiąca wraz z czterema turniejami zapowiedzianymi na marzec, kwiecień, maj i czerwiec 2022 roku. Turnieje utworzyły czteromiesięczny cykl : szachy i trzy inne gry planszowe.

Aby uprościć tworzenie tabeli "Tournaments__scores" zdefiniowaliśmy dwie funkcje pomocnicze.

```
def wyb_i(n,k):
    Lis=[]
    while len(Lis)<k:
        Lis.append(random.choice(list(range(n))))
        Lis=list(set(Lis))
        Lis.sort()
    return Lis
```

Powyższa funkcja "wyb_i" wybiera 'k' z 'n' kolejnych liczb nieujemnych.

```
def wyniki(P):
    W=[]
    for i in range(P):
        W.append(0)
    runda=1
    u=len(W)
    while u>6:
        n=0
        ind=[]
        for z in range(len(W)):
            if W[z]==0:
                n=n+1
                ind.append(z)
        k=math.floor(n/2)
        losers=wyb_i(n,k)
        for a in losers:
            W[ind[a]]=runda
        runda=runda+1
        u=n-k
    if u>4:
        n=0
        ind=[]
        for z in range(len(W)):
            if W[z]==0:
                n=n+1
                ind.append(z)
        k=n-4
        losers=wyb_i(n,k)
        for a in losers:
            W[ind[a]]=runda
        runda=runda+1
    ad=random.sample(range(4),4)
    ind=[]
    for z in range(len(W)):
        if W[z]==0:
            ind.append(z)
    for a in range(4):
        W[ind[a]]=runda+ad[a]
    return W
```

Funkcja "wyniki" generuje wyniki dla n graczy. Odrzuca ona rekurencyjnie zaokrągloną w dół połowę graczy przypisując im numer rundy, w których odpadli, aż liczba graczy spadnie poniżej 8. Następnie graczom zajmującym 4,3,2 i 1 nadajemy kolejno +1 ,+2,+3 i +4 względem wartości przypisanej graczowi na miejscu 5.

Kolejno zarówno dla turniejów szachowych jak i planszowych stworzyliśmy dwie odrębne listy graczy oraz listę uwzględniającą wszystkich graczy. Oprócz dwóch pierwszych turniejów podczas których graczy losowaliśmy wyłącznie z klientów, którzy do tego dnia dokonali jakiejś transakcji; to podczas kolejnych comiesięcznych zawodów część graczy losowaliśmy najpierw z dotychczasowych uczestników turniejów szachowych(lub planszowych), a następnie resztę z pozostałych(dotychczasowych) klientów. W celu ułatwienia tego procesu skonstruowaliśmy dwie dodatkowe funkcje mające na celu uzupełnianie tabeli "Tournaments__scores":

```
def plansz_S(starzy,razem,zakres,T_i):
    LT1=[]
    l_p=len(V_else_p)
    LT2=wyb_i(l_p,starzy)
    for p in LT2:
        LT1.append(V_else_p[p])
    while len(LT1)<razem:
        p=wyb_i(zakres,1)[0]+1
        if (p) not in V_else_p:
            V_else_p.append(p)
            V_all_p.append(p)
            LT1.append(p)
    L_W=wyniki(razem)
    for i in range(len(LT1)):
        projekt=create_db_connection("giniewicz.it", "team06","te@m0g" , "team06")
        execute_query(projekt,"""INSERT INTO tournaments__scores   ( tournament_id , player_id , customer_id , score )
    VALUES
            """ +str((T_i, V_all_p.index(LT1[i])+1,LT1[i], L_W[i]))+ """ ;""")
```

```
def szach_S(starzy,razem,zakres,T_i):
    LT1=[]
    l_p=len(V_szach_p)
    LT2=wyb_i(l_p,starzy)
    for p in LT2:
        LT1.append(V_szach_p[p])
    while len(LT1)<razem:
        p=wyb_i(zakres,1)[0]+1
        if (p) not in V_szach_p:
            V_szach_p.append(p)
            V_all_p.append(p)
            LT1.append(p)
    L_W=wyniki(razem)
    for i in range(len(LT1)):
        projekt=create_db_connection("giniewicz.it", "team06","te@m0g" , "team06")
        execute_query(projekt,"""INSERT INTO tournaments__scores   ( tournament_id , player_id , customer_id , score )
    VALUES
            """ +str((T_i, V_all_p.index(LT1[i])+1,LT1[i], L_W[i]))+ """ ;""")
```

dzięki którym sprawnie wygenerowaliśmy dane dla wszystkich kolejnych turniejów szachowych, jak i planszowych.

Bibliografia

1 https://polskienazwiska.pl/ranking/top100

2 https://www.edziecko.pl/ciaza_i_porod/7,79473,25625414,polskie-imiona-meskie-i-chlopiece-ktore-najpopularniejsze.html

3 https://8naj.pl/100-popularnych-imion/