Dokumentacja.

Spis użytych technologii:

- python,
- jupyter notebook,
- sql.

Używane biblioteki w pythonie:

- math,
- numpy,
- faker,
- pandas,
- random,
- calendar,
- dateutil.relativedelta,
- mysql.connector.

Potrzebne pliki csv oraz co zawierają:

- last_names.csv – rzeczywiste kandajyskie nazwiska,
- names_female.csv – rzeczywiste kanadyjskie damskie imiona,
- names_male.csv – rzeczywiste kanadyjskie męskie imiona,
- streets_calgary.csv – rzeczywiste ulice w Calgary,
- streets_montreal.csv – rzeczywiste ulice w Montrealu,
- streets_ottawa.csv – rzeczywiste ulice w Ottawie,
- streets_toronto.csv – rzeczywiste ulice w Toronto,
- streets_vancouver.csv – rzeczywiste ulice w Vancuver,
- real_team_names.csv – rzeczywiste nazwy drużyn hokejowych z Kanady.

Lista plików z zawartością, w kolejności uruchamiania:

1. `gen_team_name.py` – generuje nazwy druzyn naszych i przeciwników. Korzysta z real_team_names.csv. Tworzy home_team_names.csv i opponent_team_names.csv.
2. `gen_adressess.py` – dla osób i naszych placówek na podstawie miast generuje cały adres, a dla placówek przeciwników losuje adres za pomocą biblioteki faker. Korzysta z plików streets_calgary.csv, streets_montreal.csv, streets_ottawa.csv, streets_toronto.csv, streets_vancouver.csv. Tworzy addresses_people.csv i addresses_facilities.csv.
3. `gen_team_structure.py` – generuje dane dla drużyn. Korzysta z plików home_team_names.csv, opponent_team_names.csv i addresses_facilities. Tworzy home_teams_structre_v2, opponent_teams_structure_v2.csv i match_opponents_v1.csv.
4. `gen_players_dates.py` – generuje urodziny, datę dołączenia oraz datę odejścia dla graczy, dodatkowo przypisuje im numer placówki oraz kategorię wiekową. Nie korzysta z innych plików. Tworzy active_players_v2.csv i retired_players_v2.csv.
5. `connect_players_data.py` – dodaje płeć do graczy i przypisuje ich do drużyn. Korzysta z plików active_players_v2.csv i retired_players_v2.csv. Tworzy full_players_teams_v1.csv.
6. `gen_employees.py` – generuje dane dla pracowników. Nie korzysta z innych plików. Tworzy full_employees_v1.csv.
7. `connect_people.py` – łączy dane dotyczące graczy z danymi dotyczącymi pracowników. Korzysta z plików full_players_teams_v1.csv i full_employees_v1.csv. Tworzy full_people.csv.
8. `gen_people_name.py` – generuje imiona i nazwiska dla osób. Jeśli mają przypisaną płeć, dobiera odpowiednie imię. Jeśli nie mają przypisanej płci, to najpierw ją losuje. Korzysta z plików names_female.csv, names_male.csv, last_names.csv, full_people.csv. Tworzy people.csv i gender.csv.
9. `gen_ranking.py` – generuje skuteczność dla każdej z drużyn. Korzysta z match_opponents_v1.csv. Tworzy match_opponents_v2.csv.
10. `connect_addresses.py` – łączy adresy placówek oraz ludzi. Dopisuje do drużyn address_id. Korzysta z plików addresses_facilities.csv, addresses_people.csv, match_opponents_v2.csv. Tworzy match_opponents_v3.csv, addresses.csv.
11. `gen_schedule.py` – generuje przyszłe mecze (tylko na przyszły sezon). Korzysta z match_opponents_v3.csv. Tworzy schedule.csv.
12. `gen_matches.py` – generuje przeszłe mecze. Korzysta z plików match_opponents_v3.csv i full_people.csv. Tworzy full_matches.csv.
13. `gen_position.py` – zbiera pozycje graczy z każdego z meczu. Korzysta z full_matches.csv. Tworzy postions.csv.
14. `connect_matches.py` – łączy przeszłe i przyszłe mecze w jeden plik. Korzysta z plików schedule.csv i full_matches.csv. Tworzy matches.csv.
15. `gen_phone_numbers.py` – generowanie numerów telefonów dla każdej osoby. Korzysta z pliku full_people.csv. Tworzy phone_book.csv.
16. `gen_salaries.py` – generuje wypłaty dla pracowników. Początkowo tworzy plik pomocniczy reference_salaries.csv ze stałymi wypłatami dla każdej grupy pracowników na przestrzeni lat. Korzysta z pliku full_people.csv. Tworzy salaries.csv.
17. `gen_fees.py` – generuje opłaty za członkostwo wpłacone przez każdego gracza. Korzysta z pliku full_people.csv. Tworzy plik fees.csv.
18. `connect_finances.py` – łączy wszystkie przepływy finansowe dla osób. Korzysta z pliku full_people.csv, salaries.csv i fees.csv. Tworzy finances.csv.
19. `gen_equipment.py` – generuje dostępny ekwipunek w każdej z placówek. Korzysta z pliku full_people.csv. Tworzy facility.csv.
20. `select_needed_data.py` – generuje pliki csv, które zostaną wprowadzone do bazy danych.

Relacje wraz ze zbiorami zależości.

Klucze główne oznaczymy pogrubieniem.

- Tabela people: relacja people(**person_id**, first_name, last_name) ze zbiorem zależności

$$\sum = \{person\_id \rightarrow first\_name, person\_id \rightarrow last\_name, first\_name + last\_name \rightarrow person\_id\}.$$

  Id osoby mówi nam o jej imieniu, nazwisku oraz połączeniu obu. Tylko połączenie imienia i nazwiska mówi nam o id osoby, ponieważ zakładamy, że połączenie to jest unikatowe, ale imiona i nazwiska mogą się powtarzać.

- Tabela gender: relacja gender(**first_name**, gender) ze zbiorem zależności

$$\sum = \{first\_name \rightarrow gender\}.$$

  Imię jednoznacznie mówi nam o płci. Zakładamy, że nie ma takiego samego imienia dla dwóch płci.

- Tabela phone_book: relacja phone_book(**person_id**, phone) ze zbiorem zależności

$$\sum = \{person\_id \rightarrow phone, phone \rightarrow person\_id\}.$$

  Każda osoba ma unikatowy numer telefonu, dlatego zachodzi wynikanie w obie strony.

- Tabela finances: relacja finances(**person_id, date**, finantial_flow) ze zbiorem zależności

$$\sum = \{person\_id + date \rightarrow finantial\_flow\}.$$

  Ponieważ osoby dokonują płatności bądź otrzymują wypłaty co miesiąc, to tylko połączenie id osoby i daty mówi nam o wysokości danego przepływu finansowego.

- Tabela personal_info: relacja personal_info(**person_id**, team_name, position, birthdate, join_date, retire_date) ze zbiorem zależności

$$\sum = \{person\_id \rightarrow team\_name, person\_id \rightarrow position, person\_id \rightarrow birthdate, person\_id \rightarrow join\_date, person\_id \rightarrow retire\_date\}.$$

  Z id osoby jednoznacznie wynikają informacje osobowe. Natomiast żadna z tych informacji nie jest unikatowa, dlatego osobno ani w połączeniu nie pozwalają jednoznacznie zidentyfikować osoby.

- Tabela address_book: relacja address_book(**address_id**, city, street, street_number) ze zbiorem zależności

$$\sum = \{address\_id \rightarrow city, address\_id \rightarrow street, address\_id \rightarrow street\_number\}.$$

  Z id adresu wynikają miasto, ulica oraz numer domu. Jednak zakładamy, że połączenie tych trzech informacji nie musi być unikatowe - dopuszczamy współlokatorów/rodzinę.

- Tabela addresses: relacja addresses(**person_id**, address_id) ze zbiorem zależności

$$\sum = \{person\_id \rightarrow address\_id, address\_id \rightarrow person\_id\}.$$

  Zarówno id osoby jak i id adresu są unikatowe.

- Tabela facility: relacja facility(**facility_id**, address_id) ze zbiorem zależności

$$\sum = \{facility\_id \rightarrow address\_id, address\_id \rightarrow facility\_id\}.$$

  Zakładamy, że pod danym adresem znajduje się tylko jedna placówka oraz każda placówka ma tylko jeden adres.

- Tabela equipment: relacja equipment(**facility_id**, brooms, stones, shoes) ze zbiorem zależności

$$\sum = \{facility\_id \rightarrow brooms, facility\_id \rightarrow stones, facility\_id \rightarrow shoes\}.$$

  Każda placówka ma określony ekwipunek. Logiczne że zawartość schowka w każdej placówce niczego nie implikuje.

- Tabela teams: relacja teams(**team_name**, facility_id, age_category, gender_category) ze zbiorem zależności

$$\sum = \{team\_name \rightarrow facility\_id, team\_name \rightarrow age\_category, team\_name \rightarrow gender\_category\}.$$

  Każda drużyna przynależy do danej placówki oraz ma określoną kategorię. W placówkach natomiast występują różne drużyny o tych samych kategoriach, dlatego nie możemy nic z tego wywnioskować.

- Tabela matches: relacja matches(**team_name, date**, address_id, team_score, opponent_score, number_of_ends, ends_won) ze zbiorem zależności

$$\sum = \{tn + date \rightarrow address\_id, tn + date \rightarrow team\_score, tn + date \rightarrow opponent\_score, tn + date \rightarrow number\_of\_ends, tn + date \rightarrow ends\_won\},$$

  gdzie *tn* – team_name.

  Zakładamy, że każdego dnia drużyna może zagrać tylko jeden mecz. Dlatego połączenie nazwy drużyny i daty jednoznacznie określa miejsce i wynik meczu. Natomiast wyniki meczu wraz z adresami nie muszą być unikatowe.

- Tabela opponents: relacja opponents(**team_name, date**, opponent_name) ze zbiorem zależności

$$\sum = \{team\_name + date \rightarrow opponent\_name, opponent\_name + date \rightarrow team\_name\}.$$

  Dodatkowo zakładamy, że drużyna przeciwna również może grać tylko jeden mecz dziennie z naszymi drużynami, dlatego z połączenia nazwy naszej drużyny i daty możemy wywnioskować oponentów, a także z połączenia przeciwników i daty - nazwę naszej drużyny.

- Tabela positions: relacja positions(**person_id, date**, position) ze zbiorem zależności

$$\sum = \{person\_id + date \rightarrow position\}.$$

  W każdym meczu dana osoba z drużyny może mieć tylko jedną pozycję.

Uzasadnienie EKNF.

Wszędzie mamy dane atomowe. W każdej tabeli mamy zdefiniowany klucz. Wszystkie atrybuty niekluczowe zależą od całego klucza głównego. Zatem zachodzi 1NF.

Żadna kolumna niekluczowa nie jest częściowo funkcyjnie zależna od kluczy kandydujących. Zachodzi więc 2NF.

Na podstawie wypisanych relacji, możemy powiedzieć, że żaden atrybut niekluczowy nie jest zależny funkcyjnie od innych atrybutów niekluczowych. Czyli zachodzi 3NF.

Ponownie z wypisanych relacji widzimy, że wszystkie zaczynają się od całego klucza głównego albo kończą na elemencie klucza elementarnego. Dodatkowo wszystkie nasze klucze są kluczami elementarnymi, bo atrybuty elementarne kluczy kilkukolumnych nie mają żadnych zależności funkcyjnych z innymi atrubutami. Jedyną problematyczną relacją mogłoby się okazać opponents, jednak obie zależności funkcyjne zaczynają się od całego klucza głównego bądź kończą się na atrubcie klucza głównego. Ostatecznie możemy więc stwierdzić, że nasza baza jest w EKNF.

A co było najtrudniejsze? 😳

Najbardziej problematyczna okazała się normalizacja bazy danych do EKNF, ponieważ jest to wysoka postać, do której mało kto doprowadza swoją bazę, przez co ciężko było znaleźć przykłady, algorytm był dość skomplikowany, a sama normalizacja jest mało intuicyjna.

Dodatkowo ku naszemu zdziwieniu problemem okazło się również wstawienie gotowych danych do bazy. Nie działała nam komenda `LOAD DATA INFILE`, z której najczęściej korzysta się podczas wprowadzania danych z csv, dlatego musiałyśmy szukać innego rozwiązania. Ostatecznie wykorzystałyśmy pythona i bibliotekę mysql.connector.