Dokumentacja

Spis treści:

1. Spis użytych technologii
2. Lista plików i opis ich zawartości
3. Kolejność i sposób uruchamiania plików
4. Schemat projektu bazy danych
5. Zależności funkcyjne
6. EKNF
7. Najtrudniejsze podczas realizacji projektu
8. Spis użytych technologii
   1. Python w wersji 3.11.4.
   2. Biblioteki w pythonie, między innymi sqlalchemy. Bardziej szczegółowy spis użytych bibliotek w pliku requirements.txt
   3. Baza danych MariaDB, na serwerze zajęciowym
   4. ERD editor dla wizualizacji zależności między tabelami
9. Lista plików i opis ich zawartości

Pimp my wheels

|

| - data (folder z danymi dla projektu)

| \

| | - parameters (folder ze podstawowymi stałymi)

| | \

| | | - dates.json

| | |

| | | - employees.json

| | |

| | | - services\_parts.json

| |

| | - raw (folder z imionami i nazwiskami osób)

| | \

| | | - female\_first\_names.csv (imiona żeńskie)

| | |

| | | - female\_surnames.csv (nazwiska żeńskie)

| | |

| | | - male\_first\_names.csv (imiona męskie)

| | |

| | | - male\_surnames.csv (nazwiska męskie)

| |

| | - brands.csv (marki, modele i ceny samochodów)

| |

| | - equipment.csv(nazwy, typy i ceny rynkowe części samochodów)

| |

| | - female\_surnames.csv(????)

| |

| | - male\_surnames.csv(????)

| |

| | - names.csv(????)

|

| - src(folder dla podstawowej generacji bazy)

| \

| | - emulation (folder dla emulacji akcji podejmowanych przez osób)

| | \

| | | - \_\_init\_\_.py

| | |

| | | - customer\_decision\_maker.py (akcji podejmowane przez klientów)

| | |

| | | - emulation.py (emulacja jednego dnia pracy)

| | |

| | | - equipment\_generator.py (generator narzędzi)

| | |

| | | - workshop\_decision\_maker.py (akcji dotyczące napraw w warsztatu)

| | |

| | | - workshop\_emulator.py (akcji dotyczące sprzedaży/kupna warsztatu)

| |

| | - generators

| | \

| | | - \_\_init\_\_.py

| | |

| | | - brand\_geneerator.py (generator aut dla tabel)

| | |

| | | - equipment\_generator.py (generator narzędzi dla tabel)

| | |

| | | - personal\_data\_generator.py (generator danych personalnych)

| |

| | - models (folder architektur i zależności w bazie)

| | \

| | | - \_\_init\_\_.py

| | |

| | | - base.py (classa bazowa dla pozostałych tabel w sqlalchemhy)

| | |

| | | - customer.py (tabela klientów)

| | |

| | | - employee.py (tabela pracowników)

| | |

| | | - equipment.py (tabela katalogu narzędzi)

| | |

| | | - inventory.py (tabela urzytych narzędzi)

| | |

| | | - service.py (tabela napraw)

| | |

| | | - transaction.py (tabela transakcji bankowych)

| | |

| | | - vehicle.py (tabela kupionych i sprzedanych pojazdach)

| | |

| | | - workshop.py (tabela warsztatów)

| |

| | - person

| |

| | - utils

| | \

| | | - \_\_init\_\_.py

| | |

| | | - names.py (wczytywanie danych z data/raw)

| |

| | - \_\_init\_\_.py

|

| - venv (wirtualne środowisko pythona)

|

| - Dokumentacja.docx (dokumentacja)

|

| - main.py (plik generujący całą bazę)

|

| - PimpMyWheelsSchema.vuerd.json (wizualizacja zależności i tabel w bazie)

|

| - requirements.txt (spis użytych bibliotek)

1. Kolejność i sposób uruchamiania plików
   1. Na początek należy zainstalować python w wersji 3.11.4. Można to

zrobić na oficjalnej stronie:

<https://www.python.org/downloads/release/python-3114/>

Wraz z pythonem instaluje się pip (administrator bibliotek dla pythona). Można sprawdzić czy się zainstalował za pomocą wpisania polecenia:

pip list

w cmd. Jeżeli pojawia się lista zaistalowanych bibliotek, to wszystko działa, jeżeli nie, można zainstalować administrator anaconda:

<https://www.anaconda.com/download>

3.2 Należy zainstalować wszystkie biblioteki dla projektu. Można to łatwo zrobić za pomocą komendy:

pip install -r requirements.txt

Znajdując się w głównym folderze PimpMyWheels

3.3 Instalujemy Visual Studio Code. Można to zrobić z oficjalnej strony:

<https://code.visualstudio.com/download>

3.4 W Visual Studio Code instalujemy ERD Editor.

3.5 Dalej można zapoznać się z ogólną architekturą bazy w pliku PimpMyWheelsSchema.vuerd.json.

3.6 Ostatnim krokiem wchodzimy do pliku Raport.Rmd i generujemy pdf raporta za pomocą Knittr

3.7 Teraz można otworzyć Raport.pdf

1. Schemat projektu bazy danych i zależności funkcyjne

Wizualizacja schematu projektu bazy danych jest w pliku: PimpMyWheelsSchema.vuerd.json

Poniżej przedstawimy opis każdej tabeli:

* 1. Customers

id(Int, unsigned, not null, autoincrement , primary key)

name(String(25) , not null)

surname(String(25) , not null)

email(String(60) , not null)

phone\_number(String(12) , not null)

birth\_date(Date, not null)

address(String(200) , not null)

account\_creation\_date(Date, not null)

account\_deletion\_date(Date)

last\_active(Date)

* 1. Employees

id (Int, unsigned, not null, autoincrement, primary key)

name(String(25), not null)

surname(String(25), not null)

email(String(60), not null)

phone\_number(String(12), not null)

birth\_date(Date, not null)

address(String(200), not null)

workshop\_id(foreign key("workshops.id"))

position(String(100), not null)

hire\_date(Date, not null)

resignation\_date(Date)

salary(DECIMAL(8,2), unsigned, not null)

* 1. Equipment

id(Int, unsigned, not null, autoincrement, primary key)

name(String(255), not null)

type(String(50), not null)

cost(DECIMAL(8, 2), not null)

* 1. Inventory

id(Int, unsigned, not null, autoincrement, primary key)

equipment\_id(Int, unsigned, not null, foreign key(equipment.id))

service\_id(Int, unsigned, foreign key(services.id))

workshop\_id(Int, unsigned, not null, foreign key(workshops.id))

delivery\_date(Date, not null)

* 1. Services

id(Int, unsigned, not null, autoincrement, primary key)

employee\_id(Int, unsigned, foreign key(employee.id))

start\_date(Date, not null)

end\_date(Date)

work\_cost(DECIMAL(8, 2), not null)

transaction\_id(Int, foreign key(transactions.id))

description(String(100), not null)

vehicle\_id(Int, foreign key(vehicle.id))

* 1. Transactions

id(Int, unsigned, not null, autoincrement, primary key)

transaction\_method(Enum(“cash”, “card”), not null)

other\_party(Int, not null, foreign key(customers.id))

date(Date, not null)

transaction\_type(Enum(“income”, “cost”), not null)

value(DECIMAL(8, 2), not null)

* 1. Vehicles

id(Int, unsigned, not null, autoincrement, primary key)

purchase\_id(Int, unsigned, foreign key(transactions.id))

sale\_id(Int, unsigned, foreign key(transactions.id))

workshop\_id(Int, unsigned, not null , foreign key(workshops.id))

brand(String(15))

model(String(15))

* 1. Workshops

id(Int, unsigned, not null, autoincrement, primary key)

address(String(200), not null)

phone\_number(String(12), not null)

stations\_number(Int, unsigned, not null)

opening\_date(Date, not null)

1. EKNF

Wewnątrz tabel nie istnieją zależności pomiędzy kolumnami, oprócz tabeli Equipment. Skoro nazwa narzędzia definiuje jego cenę, to:

Name Cost. Algorytm Bernsteina mówi że należało by wtedy rozdzielić tabele na dwie:

1. Id, Name, Cost
2. Id, Name, Type

Skoro jest to utrudnienie struktury, stwierdziliśmy zostać przy jednej tabeli.

Pozostałe zależności funkcyjne dotyczą związków między tabelowych i zawsze mają zależność z kluczem głównym tabeli obcej. A więc cała baza nie jest EKNF, aczkolwiek mamy to przemyślone.

Spis pozostałych zależności między tabelowych:

1. workshops.id -> employees.workshop\_id
2. equipment.id -> inventory.equipment\_id
3. services.id -> inventory.service\_id
4. workshops.id -> inventory.workshop\_id
5. employees.id -> services.employee\_id
6. transactions.id -> services.transaction\_id
7. vehicles.id -> services.vehicle\_id
8. customers.id -> transactions.other\_party
9. transactions.id -> vehicles.purchase\_id
10. transactions.id -> vehicles.sale\_id
11. workshops.id -> vehicles.workshop\_id
12. Najtrudniejsze podczas realizacji projektu