Zaawansowane Bazy Danych

**Wiktor Rzońca, Jakub Ciołek**

Baza danych zasilana zewnętrznymi danymi (proces ETL), np. dla bazy danych wypożyczeń rowerów miejskich. Załadowanie, weryfikacja, korekta i uzupełnienie brakujących danych. Opracowanie zapytań dostarczających informacje statystyczne.

**1. Wstęp**

Celem projektu jest stworzenie systemu bazy danych, która będzie zasilana zewnętrznymi danymi pochodzącymi z systemu rowerów miejskich Divvy w Chicago. Projekt jest realizowany w ramach przedmiotu "Zaawansowane Bazy Danych" i obejmuje proces ETL (Extract, Transform, Load), czyli pozyskiwanie, przetwarzanie i załadowanie danych do bazy danych. Dzięki tej bazie będzie możliwe efektywne przechowywanie, analiza i generowanie zapytań statystycznych dotyczących wypożyczeń rowerów miejskich.

Projekt będzie obejmował pełny cykl pracy z danymi: od pobierania informacji z zewnętrznych źródeł, przez ich weryfikację i korektę, aż po transformację do odpowiedniego formatu, załadowanie danych do bazy i opracowanie zapytań umożliwiających analizę statystyczną. Opracowanie takich zapytań pozwoli na uzyskanie wartościowych informacji, które mogą wspomóc rozwój systemu rowerów miejskich, jak również umożliwić lepsze zarządzanie i monitorowanie stacji rowerowych.

**2. Zakres projektu**

**Pobieranie danych**

* **Pobieranie danych z zewnętrznych źródeł (API)** – automatyczne pozyskiwanie danych z API systemu rowerów Divvy.

**Weryfikacja i korekta danych**

* **Walidacja danych** – sprawdzanie spójności i poprawności danych pobranych z API.
* **Korekta błędów** – identyfikacja i poprawianie błędów, które mogą wystąpić w danych.
* **Uzupełnianie brakujących danych** – analizowanie brakujących wartości i ich uzupełnianie na podstawie dostępnych źródeł.

**Transformacja danych**

* **Przekształcenie danych do odpowiedniego formatu** – dostosowanie danych do struktury wymaganej przez bazę danych.
* **Agregacja danych** – grupowanie danych w celu ich lepszego przetwarzania i analizy.
* **Usuwanie duplikatów i niepotrzebnych danych** – oczyszczanie danych z powtarzających się rekordów oraz niepotrzebnych informacji.

**Załadowanie do bazy danych**

* **Projekt bazy danych** – zaprojektowanie struktury bazy danych, w tym tabel, relacji i indeksów.
* **Załadowanie danych do bazy** – zaimportowanie przetworzonych danych do bazy danych.

**Opracowanie zapytań statystycznych**

* + **Tworzenie zapytań SQL** – opracowanie zapytań, które pozwolą na analizę danych i uzyskanie wartościowych informacji statystycznych, takich jak liczba wypożyczeń w danym okresie, dostępność rowerów w różnych lokalizacjach czy zmiany w godzinach pracy systemu.

1. **Wybór technologii**

W projekcie wykorzystana zostanie baza danych MySQL do przechowywania i przetwarzania danych dotyczących wypożyczeń rowerów miejskich. Do komunikacji z bazą oraz obróbki danych posłuży język Python wraz z bibliotekami takimi jak pymysql i pandas, a głównym środowiskiem programistycznym będzie PyCharm. Dane będą pobierane z API, przetwarzane oraz analizowane, a wyniki zapisywane w bazie. System kontroli wersji GitHub umożliwi efektywną współpracę zespołową, śledzenie zmian w kodzie oraz zapewnienie jego wysokiej jakości.

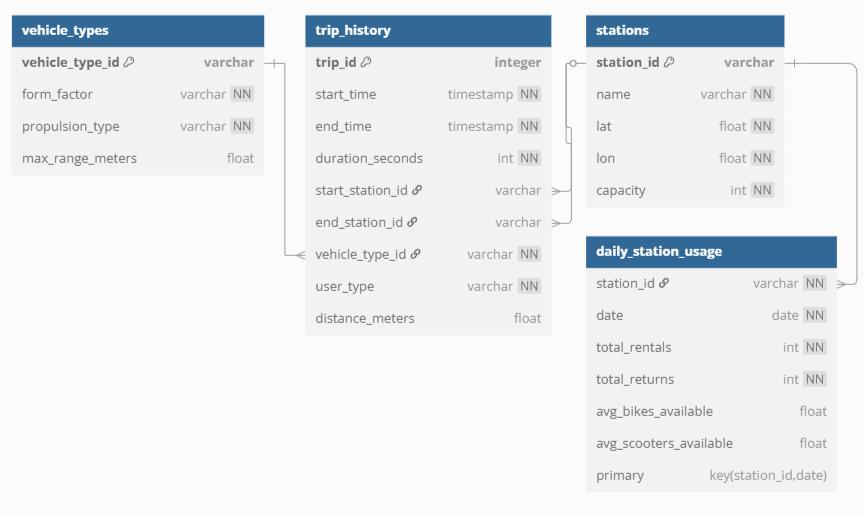
**4. Zewnętrzne dane**

Do realizacji projektu wykorzystane zostały dane udostępniane przez system rowerów miejskich Divvy w Chicago. Firma oferuje otwarte API z którego pobrane zostaną poniższe dane:

|  |  |  |
| --- | --- | --- |
| Tabela | Opis | Endpoint |
| Trip\_history | Informacje o wykonanych przejazdach | https://divvy-tripdata.s3.amazonaws.com/index.html |
| Station\_information | Informacje o stacjach z rowerami | https://gbfs.lyft.com/gbfs/2.3/chi/en/station\_information.json |
| Station\_status | Codzienna aktualizacja obecnego stanu danej stacji rowerów | https://gbfs.lyft.com/gbfs/2.3/chi/en/station\_status.json |
| Vehicle\_types | Typy dostępnych rowerów | https://gbfs.lyft.com/gbfs/2.3/chi/en/vehicle\_types.json |

**5. Diagram bazy danych**

Diagram może ulec zmianie w trakcie implementacji kodu.



**6. Opis programu**

Program składa się z trzech kluczowych segmentów:

1. Pobranie danych z API
2. Sprawdzenie poprawności danych
3. Wstawienie danych do bazy danych

**6.1. Pobranie danych z API**

Dane pobierane są z otwartego API firmy Divvy Bikes, która udostępnia darmowy dostęp do danych o stacjach i przejazdach swoich rowerów. Za tą funkcjonalność odpowiada klasa download\_data.py, która pobiera dane z endpoint’ów i zapisuje je do plików .json.

**6.2. Sprawdzenie poprawności danych**

**6.2.1 Station information**

Dane wejściowe pochodzą z pliku station\_information.json i zawierają zestaw informacji o wszystkich stacjach w systemie — w tym zarówno stacje aktywne, jak i potencjalnie nieaktywne lub testowe.

**Zakres przetwarzanych danych**

Po wczytaniu zbioru danych, kod filtruje i zachowuje tylko kluczowe kolumny:

* station\_id – unikalny identyfikator stacji,
* name – nazwa stacji,
* lat i len – współrzędne geograficzne (szerokość i długość geograficzna),
* capacity – liczba dostępnych miejsc na rowery (pojemność stacji).

**Analiza kompletności danych**

W kolejnym kroku wykonywana jest kontrola jakości danych:

* sprawdza, ile brakujących (pustych) wartości znajduje się w każdej z analizowanych kolumn (np. brak współrzędnych lub pojemności),
* oblicza liczbę unikalnych identyfikatorów stacji, aby wykryć ewentualne duplikaty w zbiorze.

**Filtracja lokalizacji geograficznych**

Następnie wykonywana jest analiza przestrzenna mająca na celu wykluczenie stacji, które znajdują się poza rzeczywistym obszarem funkcjonowania systemu – czyli poza granicami miasta Chicago i jego okolic. W tym celu zdefiniowane są granice geograficzne:

* **Szerokość geograficzna (lat)**: od 41.2 do 42.2,
* **Długość geograficzna (lon)**: od -88.2 do -87.2.

Na tej podstawie tworzony jest podzbiór stacji, które wykraczają poza ten obszar. Liczba takich przypadków jest wypisywana, co pozwala zidentyfikować dane odstające, które mogą wskazywać na błędy w lokalizacji, testowe punkty lub dane z innego regionu.

**Efekt końcowy przetwarzania**

Efektem działania kodu jest:

* przefiltrowany zbiór danych zawierający tylko najistotniejsze atrybuty stacji,
* statystyczna ocena kompletności danych,
* identyfikacja i wylistowanie stacji znajdujących się poza docelowym obszarem działania systemu.

Ten etap przygotowania danych ma kluczowe znaczenie przed dalszą analizą przestrzenną, wizualizacją na mapach czy łączeniem z danymi o przejazdach, ponieważ zapewnia jakość i spójność danych wejściowych.

**6.2.2 Station status**

Dane te odzwierciedlają aktualną dostępność rowerów, hulajnóg oraz informacje o stanie działania każdej stacji w danym momencie.

**Zakres przetwarzanych danych**

Po załadowaniu danych do programu, analizowane są trzy kluczowe atrybuty techniczne każdej stacji:

* is\_renting – czy stacja aktualnie wypożycza rowery,
* is\_returning – czy stacja przyjmuje zwroty,
* is\_installed – czy stacja jest zainstalowana i gotowa do działania.

Dla każdej z tych cech wypisywana jest liczba wystąpień poszczególnych wartości (np. 1 = tak, 0 = nie), co pozwala szybko ocenić, ile stacji jest faktycznie aktywnych i operacyjnych.

**Przetwarzanie danych o dostępności**

W kolejnym etapie skrypt wybiera istotne kolumny, które zawierają informacje o dostępnych pojazdach:

* num\_bikes\_available – liczba dostępnych klasycznych rowerów,
* num\_ebikes\_available – liczba dostępnych rowerów elektrycznych,
* num\_scooters\_available – liczba dostępnych hulajnóg.

Dane są analizowane pod kątem brakujących wartości, a tam, gdzie brakuje informacji o hulajnogach, zakłada się ich brak i uzupełnia dane wartością 0.

**Agregacja danych o dostępności**

Następnie obliczane są uśrednione wskaźniki dostępności pojazdów:

* średnia liczba dostępnych rowerów (average\_bikes\_availavle) – jako średnia z rowerów klasycznych i elektrycznych,
* liczba dostępnych hulajnóg (average\_scooters\_available) – przeniesiona z wcześniejszej kolumny.

Oryginalne kolumny dotyczące poszczególnych typów pojazdów są następnie usuwane, aby uprościć zbiór danych i zostawić tylko zagregowane informacje.

**Przygotowanie danych do dalszej analizy**

Na końcu do zestawu dodawane są nowe kolumny total\_rents i total\_returns, które początkowo przyjmują wartość 0. Mają one służyć jako miejsce do dalszego zliczania wypożyczeń i zwrotów na poziomie każdej stacji w przyszłej analizie dynamicznej.

**Efekt końcowy przetwarzania**

W wyniku działania kodu powstaje uproszczony, ale bogaty informacyjnie zestaw danych, który zawiera:

* identyfikatory stacji,
* uśrednioną dostępność rowerów i hulajnóg,
* stan gotowości operacyjnej każdej stacji,
* miejsce na dalsze metryki związane z ruchem pojazdów (np. wypożyczenia, zwroty).

Tak przygotowany zbiór może być bezpośrednio wykorzystany do tworzenia analiz statycznych lub dynamicznych (np. trendów dostępności, obciążenia stacji czy wizualizacji zasobów w czasie rzeczywistym).

**.2.3 Trip history**

Ten fragment kodu odpowiada za weryfikację i uzupełnienie danych dotyczących przejazdów rowerowych. Dane wejściowe pochodzą z pliku trip\_history.json i zawierają szczegółowe informacje o każdym przejeździe: identyfikatory stacji początkowej i końcowej, czas rozpoczęcia i zakończenia podróży, typ roweru, typ użytkownika oraz współrzędne geograficzne startu i mety.

**Zakres przetwarzanych danych**

Na początku kod wybiera tylko najważniejsze kolumny z danych przejazdów:

* informacje o czasie (started\_at, ended\_at),
* lokalizacja (start\_lat, start\_lng, end\_lat, end\_lng),
* identyfikatory stacji (start\_station\_id, end\_station\_id),
* oraz cechy przejazdu (ride\_id, rideable\_type, member\_casual).

**Weryfikacja kompletności danych**

Pierwszym etapem przetwarzania jest sprawdzenie brakujących danych. Przejazdy, które nie posiadają pełnych współrzędnych geograficznych (czyli nie mają podanej lokalizacji startowej lub końcowej), są całkowicie usuwane z dalszej analizy, ponieważ nie można ich jednoznacznie przypisać do konkretnej trasy lub stacji.

**Sprawdzenie poprawności identyfikatorów stacji**

Kolejnym krokiem jest wczytanie zbioru danych o stacjach z pliku station\_information.json. Na jego podstawie tworzony jest zbiór poprawnych identyfikatorów stacji, które rzeczywiście funkcjonują w systemie. Przejazdy zawierające błędne lub nieistniejące identyfikatory stacji są oznaczane jako niepoprawne.

**Uzupełnianie brakujących stacji na podstawie lokalizacji**

Aby uzupełnić brakujące lub niepoprawne identyfikatory stacji startowej i końcowej, wykorzystywana jest analiza przestrzenna. Tworzona jest struktura danych (KD-Tree) na podstawie współrzędnych geograficznych wszystkich dostępnych stacji, co pozwala na szybkie znalezienie najbliższej stacji względem danego punktu (startu lub końca przejazdu).

Jeśli identyfikator stacji w danym przejeździe jest nieprawidłowy, to przypisywana jest stacja znajdująca się najbliżej miejsca, z którego rozpoczęto lub zakończono podróż — na podstawie współrzędnych geograficznych.

**Efekt końcowy przetwarzania**

Efektem działania kodu jest poprawiony i uzupełniony zbiór danych o przejazdach, w którym:

* usunięto przejazdy z niekompletną lokalizacją,
* uzupełniono brakujące lub błędne identyfikatory stacji startu i końca,
* zachowano integralność danych geograficznych i logicznych.

Gotowy zbiór jest zapisywany do nowego pliku trip\_history\_fixed.json, który może być dalej wykorzystywany w analizach tras, badaniu popularności stacji lub tworzeniu wizualizacji ruchu w systemie rowerowym.

**6.3 Wstawienie danych do bazy**

Wstawianie danych do bazy MySQL realizowane jest bezpośrednio z poziomu Pythona. Poprzedzone odpowiednim przetworzeniem i oczyszczeniem informacje są przesyłane do wcześniej skonfigurowanej bazy danych, gdzie zostają zapisane w odpowiednich tabelach zgodnie z przyjętą strukturą.

**7. Napotkane trudności**

baza vehicle\_types:  
- brak wyzwań  
  
baza station\_information:  
- sprawdzono ile jest wartości null - 0  
- sprawdzono czy w bazie nie ma powtórek id stacji  
- sprawdzono czy stacje mieszczą się w chicago i obrzeżach miasta  
  
baza station\_status:  
- zostawienie potrzebnych kolumn  
- 434 wartosci null dla num\_scooters\_available, zamiana na 0  
- utworzono kolumny avg\_bikes\_available - średnia z bikes i ebikes, avg\_scooters\_available - scooter   
- potrzeba dodania kolumny total\_rents, total\_returns na podstawie tabeli trip\_history  
  
baza trip\_history:  
- pozbycie się rekordów z brakującymi szerokościami i wysokościami geograficznymi (około 25000)  
- zauważenie, że id stacji w trip\_staioons jest niezgodne z id stacji w innych bazach, zaimplementowanie przypisania stacji startowej i   
 końcowej przy pomocy KD-Tree z biblioteki scipy  
- zapisanie bazy z przypisanymi stacjami początkowymi i końcowymi do nowej bazy trip\_history\_fixed.json  
  
do zrobienia:  
- dodanie kolumny ile trwała podróż (trzeba rozwiązać sytuacje kiedy zaczynam podroz 23:50 10 marca i kończę 00:10 11 marca)   
 do dataframu w transfrom\_fixed\_trip\_history  
- dodanie distance\_meters do dataframu w transfrom\_fixed\_trip\_history  
- zsumowanie ile wypożyczono i ile zwrócono na konkretną stację do dataframu w transform\_station\_status.py

**8. Podsumowanie**