

# **System wspomagający organizację transportu pacjentów pomiędzy szpitalami**

Projektowanie baz danych

## **Projekt relacyjnej bazy danych**

### **Zespół:**

Monika Galińska

Agnieszka Kłobus

Justyna Małuszyńska

Aleksandra Stecka

# **Etap 1.**

## **Analiza świata rzeczywistego**

### **Streszczenie**

Szpitala zajmują się na co dzień ogromną liczbą pacjentów, z których każdy ma indywidualne potrzeby. W niektórych sytuacjach potrzeb danego pacjenta nie jest w stanie spełnić placówka, w której się on znajduje - nie ma w niej specjalistycznego sprzętu, narzędzi diagnostycznych, specjalistów z danej dziedziny czy też nie ma już dla niego miejsca na oddziale z powodu przepełnienia. Wtedy powstaje konieczność przewiezienia hospitalizowanego pacjenta do innej placówki. Może zaistnieć potrzeba transportu pacjenta do innego szpitala - np. aby pilnie został operowany przez tamtejszy zespół - lub do innej placówki - np. na konkretne badanie.

Przed wszystkim konieczne jest określenie, do której placówki wysłać pacjenta - trzeba wziąć pod uwagę, czy pacjent musi pojechać na badanie czy zostać przewieziony do innego szpitala, gdzie można wykonać dane badanie lub operację, czy w szpitalu, do którego ma pojechać pacjent jest dla niego dostępne miejsce. Istotną rolę pełni również odległość pomiędzy placówkami medycznymi. Kolejną rzeczą, którą należy wziąć pod uwagę jest stan pacjenta - czy pacjent wymaga miejsca w karetce czy też może być przewieziony innym środkiem transportu, czy można przetransportować go w pozycji siedzącej, czy jest samodzielny czy też należy mu zapewnić pomoc na czas podróży. Część pacjentów może być w stanie zorganizować transport we własnym zakresie.

Dzięki zebraniu danych w jednym miejscu możliwe będzie skuteczniejsze rozporządzanie zasobami, pojazdami oraz miejscami w nich, kierowcami, a także personelem medycznym. Pozwoli to również uniknąć wysłania kilku osobnych transportów w sytuacji, gdy kilku pacjentów jedzie do tej samej innej placówki.

Baza pozwoli na zebranie informacji:

- dotyczących placówek medycznych: ich lokalizacji, badań jakie są w nich wykonywane, w przypadku szpitali znajdujących się w oddziałów oraz liczby miejsc na danym oddziale,
- dotyczących potrzeb pacjentów: ich miejsca na oddziale w danym szpitalu oraz skierowań na potrzebne badania,
- dotyczących pracowników: dostępnych kierowców oraz personelu medycznego wraz z przejazdami do których zostają oddelegowani,
- dotyczących transportów: pojazdu, kierowcy i pasażerów jadących danym transportem.

Baza powinna umożliwić wyszukiwanie:

- pacjentów oczekujących na transport,
- dostępnych środków transportu,
- dostępnych kierowców i opiekunów medycznych,
- badań wykonywanych w placówkach oraz oddziałów w szpitalach,
- najbliższych dostępnych miejsc na oddziałach oraz terminów badań.

## **Cele projektu**

1. Ograniczenie liczby przejazdów między placówkami względem poprzedniego roku o 15% w ciągu roku od wprowadzenia systemu, dzięki efektywnemu przypisywaniu pacjentów do transportów (brak pustych transportów).
2. Skrócenie czasu oczekiwania na badania pacjentów o 10% względem roku poprzedniego, dzięki łatwiejszemu wyszukiwaniu wolnych terminów w placówkach sąsiadujących.
3. Utworzenie sposobu łatwego informowania kierowców oraz opiekunów medycznych o przejazdach w których wezmą udział, przez umożliwienie wszystkim dostępu do bazy danych w pierwszym wydaniu systemu.
4. Zmniejszenie o 50% ilości papierowej dokumentacji dotyczącej transportu pacjentów względem roku poprzedniego, dzięki zastąpieniu jej systemem elektronicznym.

## **Zakres projektu**

Baza danych powinna umożliwiać gromadzenie danych i planowanie przewozów pacjentów pomiędzy placówkami medycznymi znajdującymi się na terenie całej Polski. System przewidziany jest dla różnego rodzaju placówek medycznych np. szpitali, przychodni, gabinetów diagnostycznych. Należy uwzględnić potrzeby pacjentów m.in. czy wymagają stałej opieki medycznej, czy są w stanie chodzić, czy są leżący lub jeżdżą na wózku. Istotne jest też przydzielanie kierowców i opiekunów medycznych w sposób wspierający unikanie tzw. "pustych przejazdów".

Baza nie służy do przechowywania dokumentacji medycznej czy osobistej pacjentów i pracowników - system ma działać na minimalnej ilości danych wystarczających do identyfikacji danej osoby. Nie zachodzi także potrzeba przechowywania danych wszystkich pacjentów szpitali. Baza uwzględniać ma jedynie osoby, które korzystały lub będą korzystać z przewozów - za wyjątkiem pacjentów, którzy zorganizowali transport we własnym zakresie. System ma działać tylko dla Polski i uwzględniać jedynie placówki publiczne. Nie przewiduje się usuwania danych.

## **Wymagania funkcjonalne**

- W momencie upłynięcia 24h od utworzenia ostatniej kopii zapasowej, system powinien automatycznie utworzyć nową kopię.
- System powinien każdorazowo prosić o potwierdzenie przy wprowadzaniu danych przez użytkownika.
- System powinien umożliwiać dostęp do danych pacjentów wyłącznie pracownikom szpitala.
- System powinien wysyłać powiadomienie kierowcom oraz opiekunom medycznym o rezerwacji transportu.
- System powinien potwierdzać poprawną rezerwację transportu.

## **Upewnienia użtkowników**

### **Administrator:**

- logowanie i wylogowywanie
- tworzenie indywidualnych kont innych administratorów oraz kierowników

### **Kierownik:**

- logowanie i wylogowywanie
- tworzenie indywidualnych kont kierowców, lekarzy, opiekunów medycznych i pracowników administracyjnych
- wyszukiwanie, przeglądanie, dodawanie i edytowanie danych dotyczących placówek medycznych, oddziałów, badań, pacjentów, lekarzy, opiekunów medycznych oraz kierowców
- rezerwacja transportu
- wyszukiwanie i przeglądanie miejsc na oddziałach
- wyszukiwanie i przeglądanie szpitali realizujących dane badania
- wyszukiwanie i przeglądanie informacji o nadchodzących transportach
- wyszukiwanie i przeglądanie danych w historii wykonanych transportów
- wyszukiwanie, przeglądanie, dodawanie i modyfikowanie realizowanych badań

### **Kierowca:**

- logowanie i wylogowywanie
- wyszukiwanie i przeglądanie danych dotyczących transportów - dokąd, czym i kogo zawozi - oraz pojazdów
- wyszukiwanie i przeglądanie danych w historii wykonanych transportów

### **Lekarz:**

- logowanie i wylogowywanie
- wyszukiwanie, przeglądanie, dodawanie i modyfikowanie danych pacjentów
- zlecanie transportu
- wyszukiwanie, przeglądanie, dodawanie i modyfikowanie danych dotyczących skierowań
- wyszukiwanie i przeglądanie wolnych miejsc na oddziałach
- wyszukiwanie i przeglądanie placówek realizujących dane badania

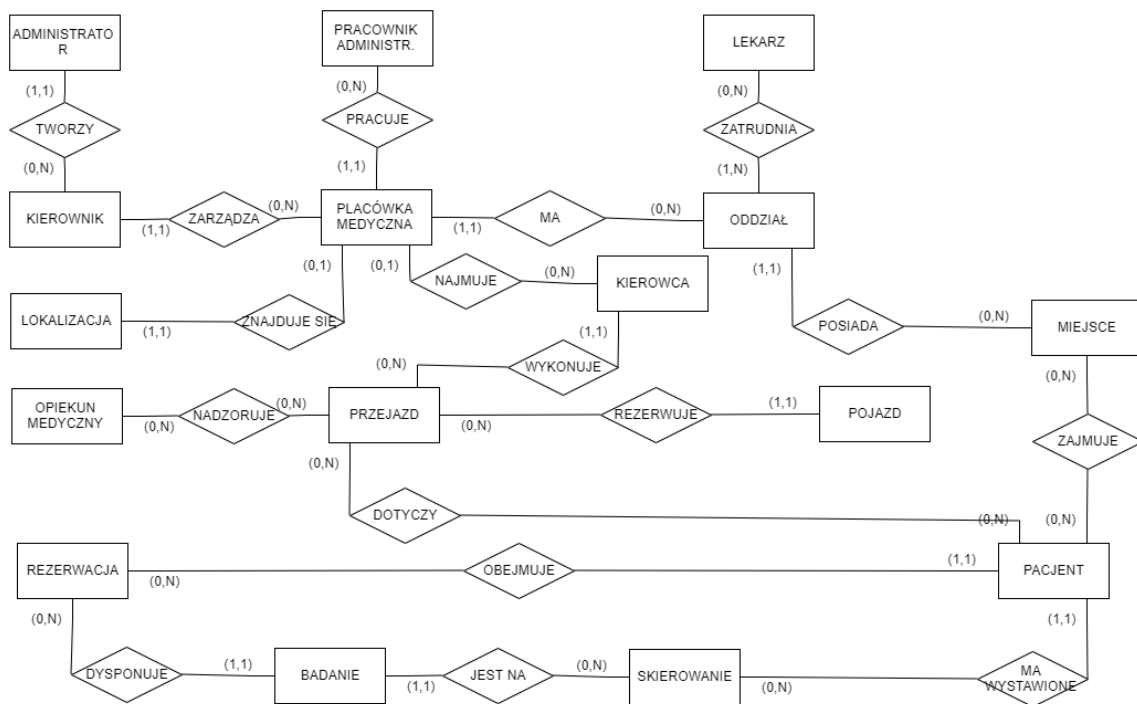
### **Opiekun medyczny:**

- logowanie i wylogowywanie
- wyszukiwanie i przeglądanie danych dotyczących pacjentów
- wyszukiwanie i przeglądanie informacji o nadchodzących transportach
- wyszukiwanie i przeglądanie danych w historii wykonanych transportów

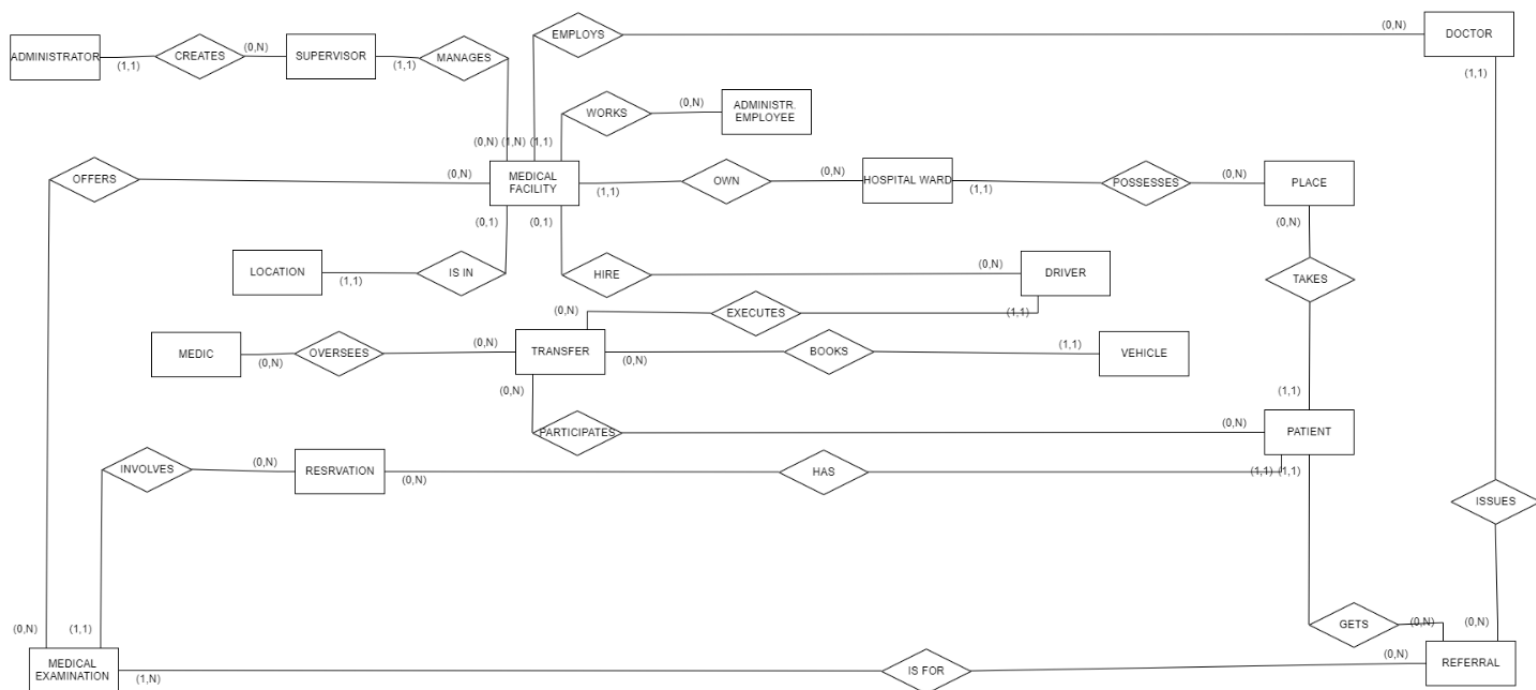
Pracownik administracyjny:

- logowanie i wylogowywanie
- tworzenie indywidualnych kont kierowców, lekarzy, opiekunów medycznych i pracowników administracyjnych
- wyszukiwanie, przeglądanie, dodawanie i modyfikowanie danych dotyczących pacjentów, lekarzy, opiekunów medycznych oraz kierowców
- rezerwacja transportu
- wyszukiwanie i przeglądanie miejsc na oddziałach
- wyszukiwanie i przeglądanie szpitali realizujących dane badania
- wyszukiwanie i przeglądanie informacji o nadchodzących transportach
- wyszukiwanie i przeglądanie danych w historii wykonanych transportów
- wyszukiwanie, przeglądanie, dodawanie i modyfikowanie realizowanych badań

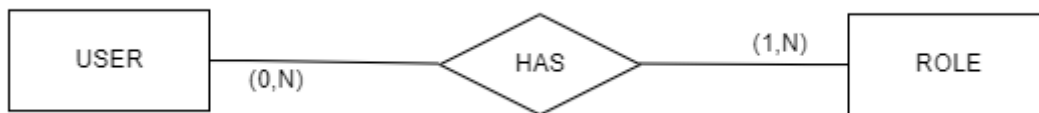
## Diagram obiektowo-związkowy



## Poprawiony diagram obiektowo-związkowy



W bazie istnieje tylko jedna encja User połączona relacją z encją Role.



Na diagramie obiektowo-związkowym pozostawiono osobne encje dla każdej roli w celu ułatwienia czytelności.

## Etap 2.

### Definicja schematów relacji i normalizacja

**Users**(Id\_user, Login, Password, Name, Surname)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_user	Identyfikator użytkownika w systemie	+
Login	Login użytkownika do systemu	+
Password	Hasło użytkownika do systemu	-
Name	Imię użytkownika	-
Surname	Nazwisko użytkownika	-

klucze kandydujące: Id\_user, Login

klucz główny: Id\_user

zależności funkcyjne:

Id\_user → Login, Password, Name, Surname

Login → Id\_user, Password, Name, Surname

**Medical\_facilities**(Id\_medical\_facility, Name, #FK\_location, #FK\_user)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_medical_facility	Identyfikator placówki medycznej w systemie	+
Name	Nazwa placówki medycznej	-
FK_location	Lokalizacja placówki medycznej – w jednej lokalizacji może znajdować się kilka placówek	-
FK_user	Identyfikator kierownika placówki medycznej	+

klucze kandydujące: Id\_medical\_facility, #FK\_user

klucz główny: Id\_medical\_facility

zależności funkcyjne:

Id\_medical\_facility → Name, #FK\_location, #FK\_user

#FK\_user → Id\_medical\_facility, Name, #FK\_location

**Exam\_offers**(Id\_exam\_offer, #FK\_medical\_facility, #FK\_medical\_exam, Daily\_limit)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_exam_offer	Identyfikator rekordu w systemie	+
FK_medical_facility	Identyfikator placówki medycznej w systemie	-
FK_medical_exam	Identyfikator badania w systemie	-
Daily_limit	Dzienny limit wykonania badania np. w przypadku rezonansu który potrzebuje odpowiedniego czasu chłodzenia	-

klucze kandydujące: {#FK\_medical\_facility, #FK\_medical\_exam}

klucz główny: {#FK\_medical\_facility, #FK\_medical\_exam}

zależności funkcyjne:

{#FK\_medical\_facility, #FK\_medical\_exam} → Daily\_limit

**Employments**(Id\_employment, #FK\_medical\_facility, #FK\_user, Employment\_date, Dimissal\_date)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_employment	Identyfikator rekordu w systemie	+
FK_medical_facility	Identyfikator placówki medycznej w systemie	-
FK_user	Identyfikator lekarza w systemie	-
Employment_date	Data zatrudnienia pracownika przez placówkę medyczną	-
Dimissal_date	Data zwolnienia pracownika przez placówkę medyczną – jest wartością opcjonalną	-

klucze kandydujące: {#FK\_medical\_facility, #FK\_user, Employment\_date}

klucz główny: {#FK\_medical\_facility, #FK\_user, Employment\_date}

zależności funkcyjne:

{#FK\_medical\_facility, #FK\_user, Employment\_date} → Dimissal\_date



**Hospital\_wards**(Id\_hospital\_ward, Name, Places, #FK\_medical\_facility)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_hospital_ward	Identyfikator oddziału w systemie	+
Name	Nazwa oddziału	-
Places	Ilość dostępnych miejsc na oddziale	-
FK_medical_facility	Identyfikator placówki medycznej, w której znajduje się oddział	-

klucze kandydujące: Id\_hospital\_ward

klucz główny: Id\_hospital\_ward

zależności funkcyjne:

Id\_hospital\_ward → Name, Places, #FK\_medical\_facility

**Locations**(Id\_location, City, Address, Postal\_code)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_location	Identyfikator lokalizacji w systemie	+
City	Nazwa miasta	-
Address	Adres w formacie Ulica Nr_domu/Nr_mieszkania	-
Postal_code	Kod pocztowy	-

klucze kandydujące: Id\_location

klucz główny: Id\_location

zależności funkcyjne:

Id\_location → City, Address, Postal\_code

**Patients**(Id\_patient, Name, Surname)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_patient	Identyfikator pacjenta w systemie	+
Name	Imię pacjenta	-
Surname	Nazwisko pacjenta	-

klucze kandydujące: Id\_patient

klucz główny: Id\_patient

zależności funkcyjne:

Id\_patient → Name, Surname

**Admissions**(Id\_admission, #FK\_hospital\_ward, #FK\_patient, Admission\_date, Discharge\_date)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_admission	Identyfikator rekordu w systemie	+
FK_hospital_ward	Identyfikator oddziału w systemie	-
FK_patient	Identyfikator pacjent w systemie	-
Admission_date	Data przyjęcia pacjenta na oddział	-
Discharge_date	Data wypisania pacjenta z oddziału – atrybut opcjonalny	-

klucze kandydujące: {#FK\_hospital\_ward, #FK\_patient, Admission\_date}

klucz główny: {#FK\_hospital\_ward, #FK\_patient, Admission\_date}

zależności funkcyjne:

{#FK\_hospital\_ward, #FK\_patient, Admission\_date} → Discharge\_date

**Transfers**(Id\_transfer, Start\_date, Start\_time, #FK\_user, #FK\_vehicle)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_transfer	Identyfikator przejazdu w systemie	+
Start_date	Data wyjazdu	-
Start_time	Godzina wyjazdu	-
FK_user	Identyfikator kierowcy w systemie	-
FK_vehicle	Identyfikator pojazdu w systemie	-

klucze kandydujące: Id\_transfer

klucz główny: Id\_transfer

zależności funkcyjne:

Id\_transfer → Start\_date, Start\_time, #FK\_user, #FK\_vehicle

**Users\_Transfers**(Id\_User\_Transfer, #FK\_user, #FK\_transfer)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_user_transfer	Identyfikator rekordu w systemie	+
FK_user	Identyfikator medyka w systemie	-
FK_transfer	Identyfikator przejazdu w systemie	-

klucze kandydujące: {#FK\_user, #FK\_transfer}

klucz główny: {#FK\_user, #FK\_transfer}

**Passengers**(Id\_passenger, #FK\_patient, #FK\_transfer, Needs\_care, Status)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_passenger	Identyfikator rekordu w systemie	+
FK_patient	Identyfikator pacjenta w systemie	-
FK_transfer	Identyfikator przejazdu w systemie	-
Needs_care	Informacja czy pacjent wymaga dodatkowej opieki – wartość Tak/Nie	-
Status	Status pacjenta, dostępne wartości: Lying, Sitting, Walking	-

klucze kandydujące: {#FK\_patient, #FK\_transfer}

klucz główny: {#FK\_patient, #FK\_transfer}

zależności funkcyjne:

{#FK\_patient, #FK\_transfer} → Needs\_care, Status

**Referrals**(Id\_referral, #FK\_user, #FK\_patient, Referral\_date, Body\_part)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_referral	Identyfikator skierowania w systemie	+
FK_user	Identyfikator lekarza w systemie	-
FK_patient	Identyfikator pacjenta w systemie	-
Referral_date	Data wystawienia skierowania	-
Body_part	Część ciała której dotyczy badanie – atrybut opcjonalny	-

klucze kandydujące: Id\_referral

klucz główny: Id\_referral

zależności funkcyjne:

Id\_referral → #FK\_user, #FK\_patient, Referral\_date, Body\_part

**Referrals\_Medical\_exams**(Id\_Ref\_Med, #FK\_referral, #FK\_medical\_exam)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_Ref_Med	Identyfikator rekordu w systemie	+
FK_referral	Identyfikator skierowania w systemie	-
FK_medical_exam	Identyfikator badania w systemie	-

klucze kandydujące: {#FK\_referral, #FK\_medical\_exam}

klucz główny: {#FK\_referral, #FK\_medical\_exam}

**Medical\_exams**(Id\_medical\_exam, Name)

Nazwa atrybutu	Znaczenie	Unikalność
Id_medical_exam	Identyfikator badania w systemie	+
Name	Nazwa badania	-

klucze kandydujące: Id\_medical\_exam

klucze główne: Id\_medical\_exam

zależności funkcyjne:

Id\_medical\_exam → Name

**Reservations**(Id\_reservation, Start\_date, Start\_time, #FK\_patient, #FK\_medical\_exam)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_reservation	Identyfikator rezerwacji w systemie	+
Start_date	Data wykonania badania	-
Start_time	Godzina wykonania badania	-
FK_patient	Identyfikator pacjenta w systemie	-
FK_medical_exam	Identyfikator badania w systemie	-

klucze kandydujące: Id\_reservation

klucze główne: Id\_reservation

zależności funkcyjne:

Id\_reservation → Start\_date, Start\_time, #FK\_patient, #FK\_medical\_exam

**Vehicles**(Id\_vehicle, Brand, Model, License\_plate, Seats)

Nazwa atrybutu	Znaczenie	Unikatowość
Id_vehicle	Identyfikator pojazdu w systemie	+
Brand	Marka pojazdu	-
Model	Model pojazdu	-
License_plate	Numer tablicy rejestracyjnej	+
Seats	Liczba miejsc w pojeździe	-

klucze kandydujące: Id\_vehicle, License\_plate

klucze główne: Id\_vehicle

zależności funkcyjne:

Id\_vehicle → Model, License\_plate, Seats

License\_plate → Id\_vehicle, Model, Seats

Roles(#FK\_user, Name)

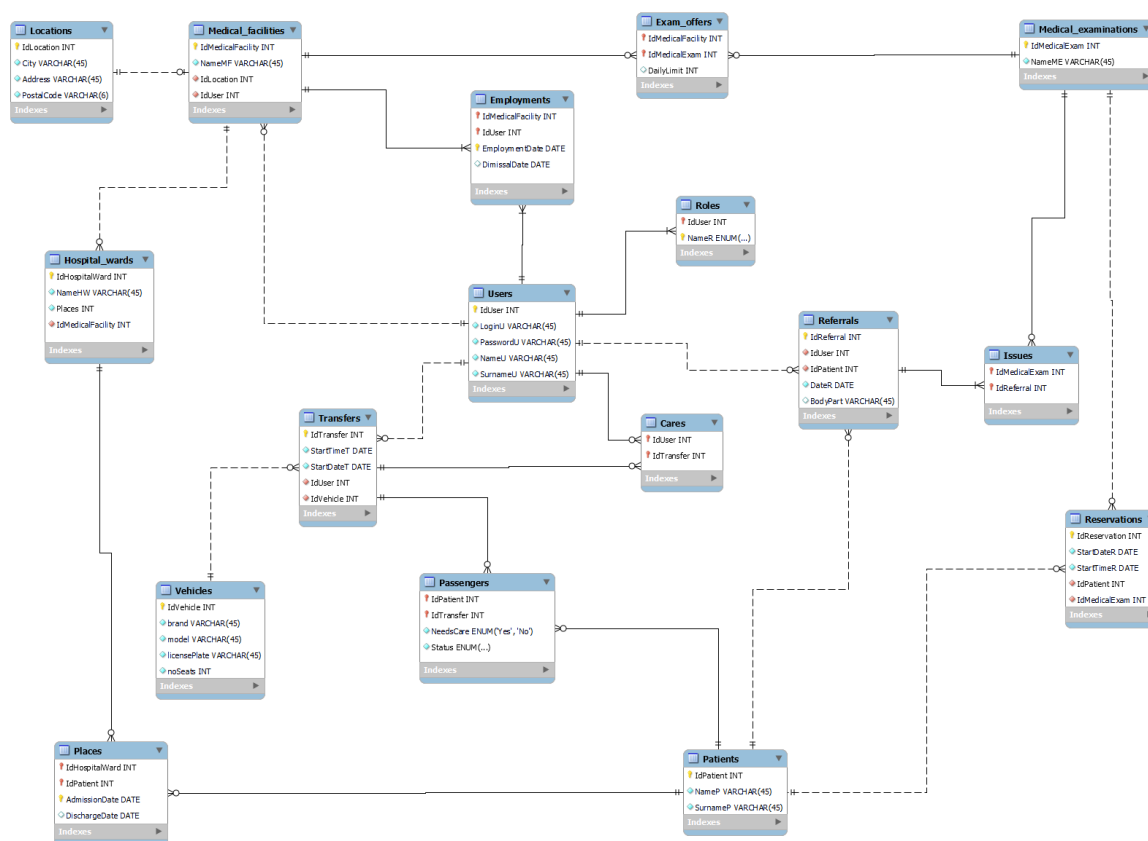
Nazwa atrybutu	Znaczenie	Unikatowość
FK_user	Identyfikator użytkownika, do którego przypisana jest rola	+
Name	Nazwa roli, dostępne wartości: Adm (Administrator), Sup (Supervisor), AEm (Administrative employee), Dri (Driver), Doc (Doctor), Med (Medic)	+

klucze kandydujące: {#FK\_user, Name}

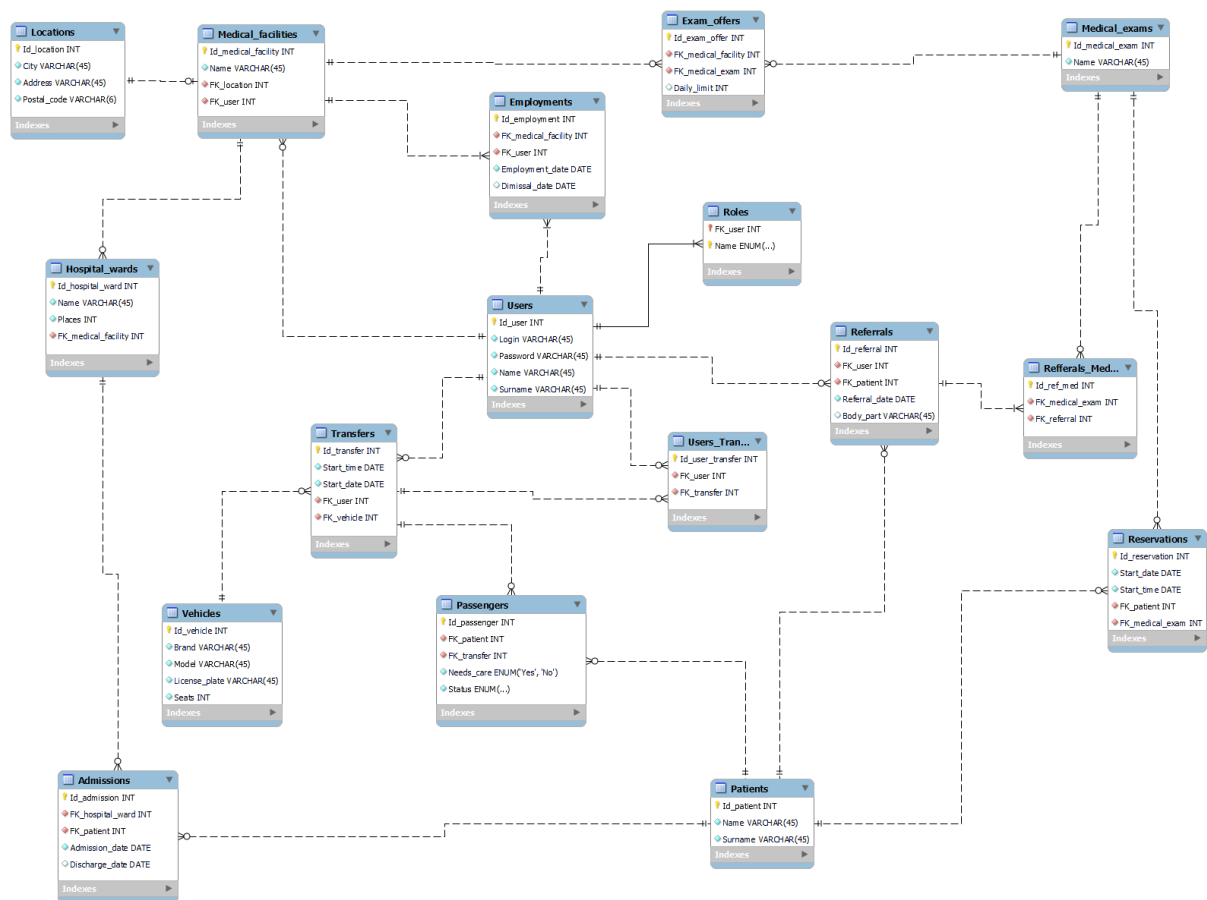
klucz główny: {#FK\_user, Name}

Nie ma zależności wielowartościowych poza trywialnymi zależnościami wielowartościowymi wynikającymi z zależności funkcyjnych.

## Diagram relacji



## Poprawiony diagram relacji



## Etap 3.

### Skrypt SQL DDL

```
-- MySQL Script generated by MySQL Workbench
-- Tue Oct 26 17:58:26 2021
-- Model: New Model      Version: 1.0
-- MySQL Workbench Forward Engineering

--
-----
-- Schema medical_transport
--
-----

drop schema medical_transport;

--
-----
-- Schema medical_transport
--
-----

CREATE SCHEMA IF NOT EXISTS `medical_transport` DEFAULT CHARACTER SET utf8 ;
USE `medical_transport` ;

--
-----
-- Table `medical_transport`.`Users`
--
-----

CREATE TABLE IF NOT EXISTS `medical_transport`.`Users` (
  `Id_user` INT NOT NULL AUTO_INCREMENT,
  `Login` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `Name` VARCHAR(45) NOT NULL,
  `Surname` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id_user`))
ENGINE = InnoDB;

--
-----
-- Table `medical_transport`.`Locations`
--
-----

CREATE TABLE IF NOT EXISTS `medical_transport`.`Locations` (
  `Id_location` INT NOT NULL AUTO_INCREMENT,
  `City` VARCHAR(45) NOT NULL,
  `Address` VARCHAR(45) NOT NULL,
  `Postal_code` VARCHAR(6) NOT NULL,
  PRIMARY KEY (`Id_location`))
ENGINE = InnoDB;

--
-----
-- Table `medical_transport`.`Medical_facilities`
--
-----

CREATE TABLE IF NOT EXISTS `medical_transport`.`Medical_facilities` (
  `Id_medical_facility` INT NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(45) NOT NULL,
  `FK_location` INT NOT NULL,
  `FK_user` INT NOT NULL,
  PRIMARY KEY (`Id_medical_facility`),
  INDEX `FK_location_idx` (`FK_location` ASC) VISIBLE,
```



```

INDEX `FK_user_idx` (`FK_user` ASC) VISIBLE,
CONSTRAINT
    FOREIGN KEY (`FK_location`)
    REFERENCES `medical_transport`.`Locations` (`Id_location`),
CONSTRAINT
    FOREIGN KEY (`FK_user`)
    REFERENCES `medical_transport`.`Users` (`Id_user`))
ENGINE = InnoDB;

-- -----
-- Table `medical_transport`.`Hospital_wards`
-- -----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Hospital_wards` (
    `Id_hospital_ward` INT NOT NULL AUTO_INCREMENT,
    `Name` VARCHAR(65) NOT NULL,
    `Places` INT NOT NULL,
    `FK_medical_facility` INT NOT NULL,
    PRIMARY KEY (`Id_hospital_ward`),
    CONSTRAINT
        FOREIGN KEY (`FK_medical_facility`)
        REFERENCES `medical_transport`.`Medical_facilities`
        (`Id_medical_facility`))
ENGINE = InnoDB;

-- -----
-- Table `medical_transport`.`Vehicles`
-- -----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Vehicles` (
    `Id_vehicle` INT NOT NULL AUTO_INCREMENT,
    `Brand` VARCHAR(45) NOT NULL,
    `Model` VARCHAR(45) NOT NULL,
    `License_plate` VARCHAR(45) NOT NULL,
    `Seats` INT NOT NULL,
    `FK_medical_facility` INT NOT NULL,
    PRIMARY KEY (`Id_vehicle`),
    CONSTRAINT CHECK(`Seats`>=0),
    CONSTRAINT `FK_medical_facility`
        FOREIGN KEY (`FK_medical_facility`)
        REFERENCES `medical_transport`.`Medical_facilities`
        (`Id_medical_facility`))
ENGINE = InnoDB;

-- -----
-- Table `medical_transport`.`Transfers`
-- -----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Transfers` (
    `Id_transfer` INT NOT NULL AUTO_INCREMENT,
    `Start_time` TIME NOT NULL,
    `Start_date` DATE NOT NULL,
    `FK_user` INT NOT NULL,
    `FK_vehicle` INT NOT NULL,

```

```

`FK_facility_from` INT NOT NULL,
`FK_facility_to` INT NOT NULL,
PRIMARY KEY (`Id_transfer`),
CONSTRAINT
    FOREIGN KEY (`FK_user`)
    REFERENCES `medical_transport`.`Users` (`Id_user`),
CONSTRAINT
    FOREIGN KEY (`FK_vehicle`)
    REFERENCES `medical_transport`.`Vehicles` (`Id_vehicle`),
CONSTRAINT `FK_facility_from`
    FOREIGN KEY (`FK_facility_from`)
    REFERENCES `medical_transport`.`Medical_facilities`
    (`Id_medical_facility`),
CONSTRAINT `FK_facility_to`
    FOREIGN KEY (`FK_facility_to`)
    REFERENCES `medical_transport`.`Medical_facilities`
    (`Id_medical_facility`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Patients`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Patients` (
  `Id_patient` INT NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(45) NOT NULL,
  `Surname` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id_patient`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Admissions`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Admissions` (
  `Id_admission` INT NOT NULL AUTO_INCREMENT,
  `FK_hospital_ward` INT NOT NULL,
  `FK_patient` INT NOT NULL,
  `Admission_date` DATE NOT NULL,
  `Discharge_date` DATE,
  PRIMARY KEY (`Id_admission`),
  CONSTRAINT CHECK(`Admission_date`<=`Discharge_date`),
  CONSTRAINT
    FOREIGN KEY (`FK_hospital_ward`)
    REFERENCES `medical_transport`.`Hospital_wards` (`Id_hospital_ward`),
  CONSTRAINT
    FOREIGN KEY (`FK_patient`)
    REFERENCES `medical_transport`.`Patients` (`Id_patient`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Referrals`
-- -----

```

```

-----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Referrals` (
  `Id_referral` INT NOT NULL AUTO_INCREMENT,
  `FK_user` INT NOT NULL,
  `FK_patient` INT NOT NULL,
  `Referral_date` DATE NOT NULL,
  `Body_part` VARCHAR(45) NULL,
  PRIMARY KEY (`Id_referral`),
  CONSTRAINT
    FOREIGN KEY (`FK_user`)
      REFERENCES `medical_transport`.`Users` (`Id_user`),
  CONSTRAINT
    FOREIGN KEY (`FK_patient`)
      REFERENCES `medical_transport`.`Patients` (`Id_patient`))
ENGINE = InnoDB;

-----
-- Table `medical_transport`.`Medical_exams`
-----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Medical_exams` (
  `Id_medical_exam` INT NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id_medical_exam`))
ENGINE = InnoDB;

-----
-- Table `medical_transport`.`Reservations`
-----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Reservations` (
  `Id_reservation` INT NOT NULL AUTO_INCREMENT,
  `Start_date` DATE NOT NULL,
  `Start_time` TIME NOT NULL,
  `FK_patient` INT NOT NULL,
  `FK_medical_exam` INT NOT NULL,
  PRIMARY KEY (`Id_reservation`),
  CONSTRAINT
    FOREIGN KEY (`FK_patient`)
      REFERENCES `medical_transport`.`Patients` (`Id_patient`),
  CONSTRAINT
    FOREIGN KEY (`FK_medical_exam`)
      REFERENCES `medical_transport`.`Medical_exams` (`Id_medical_exam`))
ENGINE = InnoDB;

-----
-- Table `medical_transport`.`Roles`
-----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Roles` (
  `Id_role` INT NOT NULL AUTO_INCREMENT,
  `FK_user` INT NOT NULL,
  `Name` ENUM("Adm", "Sup", "AEm", "Dri", "Doc", "Med") NOT NULL,

```

```

PRIMARY KEY (`Id_role`),
CONSTRAINT
    FOREIGN KEY (`FK_user`)
    REFERENCES `medical_transport`.`Users` (`Id_user`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Exam_offers`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Exam_offers` (
  `Id_exam_offer` INT NOT NULL AUTO_INCREMENT,
  `FK_medical_facility` INT NOT NULL,
  `FK_medical_exam` INT NOT NULL,
  `Daily_limit` INT NULL,
  PRIMARY KEY (`Id_exam_offer`),
  CONSTRAINT CHECK(`Daily_limit`>=0),
  CONSTRAINT
    FOREIGN KEY (`FK_medical_facility`)
    REFERENCES `medical_transport`.`Medical_facilities`
    (`Id_medical_facility`),
  CONSTRAINT
    FOREIGN KEY (`FK_medical_exam`)
    REFERENCES `medical_transport`.`Medical_exams` (`Id_medical_exam`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Employments`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Employments` (
  `Id_employment` INT NOT NULL AUTO_INCREMENT,
  `FK_medical_facility` INT NOT NULL,
  `FK_user` INT NOT NULL,
  `Employment_date` DATE NOT NULL,
  `Dismissal_date` DATE NULL,
  PRIMARY KEY (`Id_employment`),
  CONSTRAINT CHECK(`Employment_date`<=`Dismissal_date`),
  CONSTRAINT
    FOREIGN KEY (`FK_medical_facility`)
    REFERENCES `medical_transport`.`Medical_facilities`
    (`Id_medical_facility`),
  CONSTRAINT
    FOREIGN KEY (`FK_user`)
    REFERENCES `medical_transport`.`Users` (`Id_user`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Passengers`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Passengers` (
  `Id_passenger` INT NOT NULL AUTO_INCREMENT,

```

```

`FK_patient` INT NOT NULL,
`FK_transfer` INT NOT NULL,
`Needs_care` ENUM('Yes', 'No') NOT NULL,
`Status` ENUM('Lying', 'Sitting', 'Walking') NOT NULL,
PRIMARY KEY (`Id_passenger`),
CONSTRAINT
    FOREIGN KEY (`FK_patient`)
    REFERENCES `medical_transport`.`Patients` (`Id_patient`),
CONSTRAINT
    FOREIGN KEY (`FK_transfer`)
    REFERENCES `medical_transport`.`Transfers` (`Id_transfer`)
    ON DELETE CASCADE)
ENGINE = InnoDB;

-- -----
-- Table `medical_transport`.`Refferals_Medical_exams`
-- -----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Refferals_Medical_exams` (
    `Id_ref_med` INT NOT NULL AUTO_INCREMENT,
    `FK_medical_exam` INT NOT NULL,
    `FK_referral` INT NOT NULL,
    PRIMARY KEY (`Id_ref_med`),
    CONSTRAINT
        FOREIGN KEY (`FK_medical_exam`)
        REFERENCES `medical_transport`.`Medical_exams` (`Id_medical_exam`),
    CONSTRAINT
        FOREIGN KEY (`FK_referral`)
        REFERENCES `medical_transport`.`Referrals` (`Id_referral`))
ENGINE = InnoDB;

-- -----
-- Table `medical_transport`.`Users_Transfers`
-- -----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Users_Transfers` (
    `Id_user_transfer` INT NOT NULL AUTO_INCREMENT,
    `FK_user` INT NOT NULL,
    `FK_transfer` INT NOT NULL,
    PRIMARY KEY (`Id_user_transfer`),
    CONSTRAINT
        FOREIGN KEY (`FK_user`)
        REFERENCES `medical_transport`.`Users` (`Id_user`),
    CONSTRAINT medical_facilities
        FOREIGN KEY (`FK_transfer`)
        REFERENCES `medical_transport`.`Transfers` (`Id_transfer`)
        ON DELETE CASCADE)
ENGINE = InnoDB;

```

## Wykorzystane instrukcje

Wykorzystane instrukcje:

- CREATE SCHEMA - tworzy schemat struktury bazy danych, w którym następnie tworzone będą relacje,
- DEFAULT CHARACTER SET - określa wykorzystywany system kodowania,
- USE - określa schemat struktury bazy danych, do którego odnoszą się następne instrukcje,
- CREATE TABLE -
  - o tworzy relację o podanej nazwie i atrybutach podanych w nawiasie i oddzielonych przecinkami,
  - o w nawiasie należy podać wszystkie atrybuty wraz z ich dziedziną, można także ustalić, że każda wartość atrybutu musi być niepusta oraz ustalić atrybut na autonumerowanie,
  - o pozwala zdefiniować klucz główny relacji słowem kluczowym PRIMARY KEY,
  - o pozwala zdefiniować atrybuty jako indeksy słowem kluczowym INDEX - indeksy są niewidzialne dla użytkowników i używane aby przyspieszyć wyszukiwanie danych,
  - o słowo kluczowe UNIQUE dopisane przed INDEX pozwala zdefiniować atrybuty jako indeksy bez duplikatów,
  - o słowo kluczowe VISIBLE określa atrybuty zdefiniowane jako indeksy jako dostępne dla optymalizatora kwerend. Indeksy są domyślnie określane jako VISIBLE.
- ENGINE - określa silnik pamięci masowej dla MySQL,
- CONSTRAINT -
  - o ograniczenie dla danych wstawianych do tabeli,
  - o pozwala zdefiniować klucz obcy za pomocą słowa kluczowego FOREIGN KEY oraz określić, z której relacji pochodzi klucz za pomocą słowa kluczowego REFERENCES - takie ograniczenie zatrzymuje akcje, które mogłyby spowodować złamanie powiązań między relacjami (więzy integralności),
  - o pozwala zdefiniować wywołania kaskadowe słowami kluczowymi ON DELETE oraz ON UPDATE,
  - o często wykorzystywane z instrukcją CHECK, która dodatkowo ogranicza dziedzinę atrybutów - CHECK przyjmuje jako argument warunek, który musi być spełniony dla wszystkich krotek w relacji.

## **Więzy integralności, typy wywołań kaskadowych, instrukcja CHECK**

Więzy integralności służą ochronie powiązań między relacjami przed błędami - na przykład wprowadzeniem jako klucza obcego wartości, której nie ma w relacji, z której ten klucz obcy

pochodzi. Sprawdzanie spójności połączeń między relacjami dokonuje się definiując odpowiednie ograniczenie:

```
CONSTRAINT
    FOREIGN KEY (`nazwa_klucza_obcego`)
    REFERENCES `nazwa_schematu`.`nazwa_relacji` (`nazwa_klucza`),
```

Wywołania kaskadowe dzielimy na:

- aktualizację danych,
- usuwanie danych.

W projekcie występuje kaskadowe usuwanie danych w przypadku powiązania transportu z pasażerem, jeśli usuniemy dany transport, to nie ma sensu przechowywanie danych powiązanych z nim pasażerów. Analogicznie wygląda sytuacja dla tabeli “Users\_Transfers”, ponieważ usuwając transport chcemy usunąć także jego powiązania z przypisanymi opiekunami medycznymi.

W bazie zabroniona jest zmiana Id zatem wywołania dotyczące aktualizacji nie są potrzebne. Założono również przechowywanie danych archiwalnych, więc poza wyżej określonym przypadkiem nie występuje usuwanie.

Instrukcję “CHECK” wykorzystano w projekcie głównie do zapewnienia ograniczenia możliwości wprowadzania dat i liczb. Za jej pomocą sprawdzamy czy data wypisu jest nie wcześniejsza od daty przyjęcia pacjenta oraz zapewniamy, że np. pojazdy do przewozu pacjentów i oddziały szpitalne nie mogą mieć ujemnej liczby miejsc. “CHECK” służy do sprawdzania poprawności danych jeszcze przed wprowadzeniem ich do bazy.

## **Etap 4.**

Wykorzystano język Python z uwagi na jego prostotę oraz dostępne biblioteki do generowania pseudolosowych wartości na przykład imion, nazwisk, dat, loginów czy numerów rejestracyjnych.

Wykorzystano bibliotekę SQLAlchemy do nawiązania połączenia z bazą danych.

### **Skrypt w języku Python**

#### **Plik constants.py:**

```
import datetime

START_DATE = datetime.date(2021, 1, 1)
END_DATE = datetime.date(2021, 11, 1)
EMPLOYMENT_DATE = datetime.date(1950, 1, 1)

SEATS = [1, 4, 5, 6, 7, 8, 9, 10, 13, 17, 18, 19, 20, 24]

ROLES = ["Adm", "Sup", "AEm", "Dri", "Doc", "Med"]

NEEDS_CARE = ["Yes", "No"]

STATUS = ["Lying", "Sitting", "Walking"]

BODY_PARTS = ["head", "spine", "arm", "hand", "finger", "leg", "knee",
"foot", "toe", "stomach", "shoulder", "eye",
"ear", "breast", "prostate", "heart", "lung", "hip", "pelvis",
"appendix", "bladder", "liver", "kidney",
"chest"]

HOSPITAL_WARDS = ["Oddział Anestezjologii i Intensywnej Terapii", "Oddział
Chirurgii Ogólnej", "Oddział Dzieciecy",
"Oddział Chirurgii Ogólnej i Onkologicznej", "Oddział
Chirurgii Urazowo – Ortopedycznej",
"Oddział Chorob Płuc", "Oddział Chemioterapii", "Oddział
Chorob Wewnętrznych", "Blok operacyjny",
"Oddział Kardiologiczny", "Oddział Nefrologiczny", "Oddział
Chorob Wewnętrznych",
"Oddział Neonatologiczny", "Oddział Neurologiczny",
"Oddział Pediatriczny", "Oddział Nefrologii",
"Oddział Położniczo – Ginekologiczny", "Oddział
Psychiatryczny", "Oddział Endokrynologii",
"Oddział Rehabilitacji Neurologicznej", "Oddział
Rehabilitacji Kardiologicznej",
"Oddział Rehabilitacyjny Ogólny", "Oddział Udarowy",
"Oddział Urologiczny", "Stacja Dializ",
"Szpitalny Oddział Ratunkowy", "Oddział Kardiochirurgii",
"Oddział Obserwacyjno – Zakazny"]
```



```

        "Oddzial Dzienny Psychiatryczny", "Oddzial Chirurgii
Plastycznej", "Oddzial Neurochirurgiczny",
        "Oddzial Okulistyczny", "Oddzial Psychiatryczny", "Oddzial
Leczenia Uzaleznien",
        "Oddzial Rehabilitacyjny", "Oddzial Chirurgii Naczyniowej",
"Oddzial Nadciśnienia Tetniczego",
        "Oddzial Gastroenterologii", "Oddzial Laryngologiczny",
"Oddzial Onkologii Klinicznej",
        "Oddzial Reumatologiczny", "Oddzial Otolaryngologiczny",
"Oddzial Geriatryczny"]

```

```

USERS_LIMIT = 6000
LOCATIONS_LIMIT = 1250
MEDICAL_FACILITIES_LIMIT = 500
HOSPITAL_WARDS_LIMIT = 500
VEHICLES_LIMIT = 1000
TRANSFERS_LIMIT = 7000
PATIENTS_LIMIT = 7500
ADMISSIONS_LIMIT = 8500
REFERRALS_LIMIT = 9000
MEDICAL_EXAMS_LIMIT = 175
RESERVATIONS_LIMIT = 600
ROLES_LIMIT = 4500
EXAM_OFFERS_LIMIT = 2300
EMPLOYMENTS_LIMIT = 800
PASSENGERS_LIMIT = 8750
REFERRALS_MEDICAL_EXAMS = 3500
USERS_TRANSFERS_LIMIT = 4500

```

```

DAILY_LIMIT_LIMIT = 75
PLACES_IN_WARD_LIMIT = 80
POSTAL_CODE_LOWER_LIMIT = 10000
POSTAL_CODE_UPPER_LIMIT = 99999

```

## Plik tables.py:

```

from sqlalchemy import Column, Integer, String, Date, Time
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship, declarative_base

```

```

Base = declarative_base()

```

```

class Users(Base):
    __tablename__ = 'Users'

    Id_user = Column(Integer, primary_key=True)
    Login = Column(String)
    Password = Column(String)
    Name = Column(String)
    Surname = Column(String)

    roles = relationship('Roles', backref='users')

```

```

        medical_facilities = relationship('Medical_facilities',
backref='users')
        transfers = relationship('Transfers', backref='users')
        referrals = relationship('Referrals', backref='users')
        employments = relationship('Employments', backref='users')
        users_transfers = relationship('Users_Transfers', backref='users')

class Locations(Base):
    __tablename__ = 'Locations'

    Id_location = Column(Integer, primary_key=True, autoincrement=True)
    City = Column(String)
    Address = Column(String)
    Postal_code = Column(String)

    medical_facilities = relationship('Medical_facilities',
backref='locations')

class Medical_facilities(Base):
    __tablename__ = 'Medical_facilities'

    Id_medical_facility = Column(Integer, primary_key=True,
autoincrement=True)
    Name = Column(String)
    FK_location = Column(Integer, ForeignKey('Locations.Id_location'))
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))

    hospital_wards = relationship('Hospital_wards',
backref='medical_facilities')
    employments = relationship('Employments', backref='medical_facilities')
    exam_offers = relationship('Exam_offers', backref='medical_facilities')
    transfers = relationship('Transfers', backref='medical_facilities')
    vehicles = relationship('Vehicles', backref='medical_facilities')

class Hospital_wards(Base):
    __tablename__ = 'Hospital_wards'

    Id_hospital_ward = Column(Integer, primary_key=True,
autoincrement=True)
    Name = Column(String)
    Places = Column(Integer)
    FK_medical_facility = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))

    admissions = relationship('Admissions', backref='hospital_wards')

class Vehicles(Base):
    __tablename__ = 'Vehicles'

```

```

    Id_vehicle = Column(Integer, primary_key=True, autoincrement=True)
    Brand = Column(String)
    Model = Column(String)
    License_plate = Column(String)
    Seats = Column(Integer)
    FK_medical_facility = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))

    transfers = relationship('Transfers', backref='vehicles')

class Transfers(Base):
    __tablename__ = 'Transfers'

    Id_transfer = Column(Integer, primary_key=True, autoincrement=True)
    Start_time = Column(Time)
    Start_date = Column(Date)
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))
    FK_vehicle = Column(Integer, ForeignKey('Vehicles.Id_vehicle'))
    FK_facility_from = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))
    FK_facility_to = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))

    passengers = relationship('Passengers', backref='transfers')
    users_transfers = relationship('Users_Transfers', backref='transfers')

class Roles(Base):
    __tablename__ = 'Roles'

    Id_role = Column(Integer, primary_key=True, autoincrement=True)
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))
    Name = Column(String)
    unique_together = ('FK_user', 'Name')

class Patients(Base):
    __tablename__ = 'Patients'

    Id_patient = Column(Integer, primary_key=True, autoincrement=True)
    Name = Column(String)
    Surname = Column(String)

    reservations = relationship('Reservations', backref='patients')
    admissions = relationship('Admissions', backref='patients')
    referrals = relationship('Referrals', backref='patients')
    passengers = relationship('Passengers', backref='patients')

class Admissions(Base):
    __tablename__ = 'Admissions'

```

```

        Id_admission = Column(Integer, primary_key=True, autoincrement=True)
        FK_hospital_ward = Column(Integer,
ForeignKey('Hospital_wards.Id_hospital_ward'))
        FK_patient = Column(Integer, ForeignKey('Patients.Id_patient'))
        Admission_date = Column(Date)
        Discharge_date = Column(Date, nullable=True)

class Referrals(Base):
    __tablename__ = 'Referrals'

    Id_referral = Column(Integer, primary_key=True, autoincrement=True)
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))
    FK_patient = Column(Integer, ForeignKey('Patients.Id_patient'))
    Referral_date = Column(Date)
    Body_part = Column(String, nullable=True)

    referrals_medical_exams = relationship('Referrals_Medical_exams',
backref='referrals')

class Medical_exams(Base):
    __tablename__ = 'Medical_exams'

    Id_medical_exam = Column(Integer, primary_key=True, autoincrement=True)
    Name = Column(String)

    reservations = relationship('Reservations', backref='medical_exams')
    exam_offers = relationship('Exam_offers', backref='medical_exams')
    referrals_medical_exams = relationship('Referrals_Medical_exams',
backref='medical_exams')

class Reservations(Base):
    __tablename__ = 'Reservations'

    Id_reservaion = Column(Integer, primary_key=True, autoincrement=True)
    Start_date = Column(Date)
    Start_time = Column(Time)
    FK_patient = Column(Integer, ForeignKey('Patients.Id_patient'))
    FK_medical_exam = Column(Integer,
ForeignKey('Medical_exams.Id_medical_exam'))

class Exam_offers(Base):
    __tablename__ = 'Exam_offers'

    Id_exam_offers = Column(Integer, primary_key=True, autoincrement=True)
    FK_medical_facility = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))
    FK_medical_exam = Column(Integer,
ForeignKey('Medical_exams.Id_medical_exam'))
    Daily_limit = Column(Integer, nullable=True)

```

```

class Employments(Base):
    __tablename__ = 'Employments'

    Id_employment = Column(Integer, primary_key=True, autoincrement=True)
    FK_medical_facility = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))
    Employment_date = Column(Date)
    Dismissal_date = Column(Date, nullable=True)

class Passengers(Base):
    __tablename__ = 'Passengers'

    Id_passenger = Column(Integer, primary_key=True, autoincrement=True)
    FK_patient = Column(Integer, ForeignKey('Patients.Id_patient'))
    FK_transfer = Column(Integer, ForeignKey('Transfers.Id_transfer'))
    Needs_care = Column(String)
    Status = Column(String)

class Referrals_Medical_exams(Base):
    __tablename__ = 'Refferals_Medical_exams'

    Id_ref_med = Column(Integer, primary_key=True, autoincrement=True)
    FK_medical_exam = Column(Integer,
ForeignKey('Medical_exams.Id_medical_exam'))
    FK_referral = Column(Integer, ForeignKey('Referrals.Id_referral'))

class Users_Transfers(Base):
    __tablename__ = 'Users_Transfers'

    Id_user_transfer = Column(Integer, primary_key=True,
autoincrement=True)
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))
    FK_transfer = Column(Integer, ForeignKey('Transfers.Id_transfer'))

```

## **Plik main.py:**

```

from random import *

from faker import Faker
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from sqlalchemy.sql import text

from constants import *
from tables import *

fake = Faker()

```

```

user = 'root'
pwd = 'root123'
host = 'localhost'
db = 'medical_transport'
mysql_engine = create_engine('mysql://{0}:{1}@{2}/{3}'.format(user, pwd,
host, db))

Session = sessionmaker(bind=mysql_engine)
session = Session()

def get_logins(session):
    return [user.Login for user in session.query(Users).all()]

def get_supervisors(session):
    return [medical_facility.FK_user for medical_facility in
session.query(Medical_facilities).all()]

def get_license_plates(session):
    return [vehicle.License_plate for vehicle in
session.query(Vehicles).all()]

def get_hospital_wards(session):
    return [(hospital_ward.Name, hospital_ward.FK_medical_facility)
            for hospital_ward in session.query(Hospital_wards).all()]

def get_users_with_role(session, role_id):
    users = session.query(Roles).all()
    users_with_role = []
    for user in users:
        if user.Name == ROLES[role_id]:
            users_with_role.append(user.FK_user)
    return users_with_role

def generate_users(session):
    logins = get_logins(session)
    for i in range(0, USERS_LIMIT):
        login = fake.user_name()
        while logins.count(login) != 0:
            login = fake.user_name()
        logins.append(login)
        new_user = Users(
            Login=login,
            Password=fake.password(),
            Name=fake.first_name(),
            Surname=fake.last_name()
        )

```

```

session.add(new_user)
session.commit()

def generate_roles(session):
    user_index = 1
    for i in range(0, MEDICAL_FACILITIES_LIMIT):
        role = Roles(
            FK_user=user_index,
            Name=ROLES[1]
        )
        user_index = user_index + 1
        session.add(role)
    for i in range(0, min(USERS_LIMIT - MEDICAL_FACILITIES_LIMIT,
ROLES_LIMIT)):
        role = Roles(
            FK_user=user_index,
            Name=choice(ROLES)
        )
        user_index = user_index + 1
        session.add(role)
    for i in range(0, ROLES_LIMIT - USERS_LIMIT):
        role = Roles(
            FK_user=randint(1, USERS_LIMIT),
            Name=choice(ROLES)
        )
        session.add(role)
    session.commit()

def generate_locations(session):
    for i in range(0, LOCATIONS_LIMIT):
        location = Locations(
            City=fake.city(),
            Address=fake.street_address(),
            Postal_code=randint(POSTAL_CODE_LOWER_LIMIT,
POSTAL_CODE_UPPER_LIMIT)
        )
        session.add(location)
    session.commit()

def generate_medical_facilities(session):
    supervisors = get_supervisors(session)
    possible_supervisors = get_users_with_role(session, 1)
    for i in range(0, MEDICAL_FACILITIES_LIMIT):
        supervisor = choice(possible_supervisors)
        while supervisors.count(supervisor) != 0:
            supervisor = choice(possible_supervisors)
        supervisors.append(supervisor)
    medical_facility = Medical_facilities(
        Name=fake.pystr(),
        FK_location=randint(1, MEDICAL_FACILITIES_LIMIT),

```

```

        FK_user=supervisor
    )
    session.add(medical_facility)
    session.commit()

def generate_hospital_wards(session):
    hospital_wards = []
    for i in range(0, HOSPITAL_WARDS_LIMIT):
        medical_facility = randint(1, MEDICAL_FACILITIES_LIMIT)
        hospital_ward = choice(HOSPITAL_WARDS)
        while hospital_wards.count([hospital_ward, medical_facility]) != 0:
            hospital_ward = choice(HOSPITAL_WARDS)
        hospital_ward_A = Hospital_wards(
            Name=hospital_ward,
            Places=randint(0, PLACES_IN_WARD_LIMIT),
            FK_medical_facility=medical_facility
        )
        hospital_wards.append([hospital_ward, medical_facility])
    session.add(hospital_ward_A)
    session.commit()

def generate_vehicles(session):
    license_plates = get_license_plates(session)
    for i in range(0, VEHICLES_LIMIT):
        license_plate = fake.license_plate()
        while license_plates.count(license_plate) != 0:
            license_plate = fake.license_plate()
        license_plates.append(license_plate)
    seats = int(choice(SEATS))
    vehicle = Vehicles(
        Brand=fake.pystr(2, 20),
        Model=fake.pystr(3, 20),
        License_plate=license_plate,
        Seats=seats,
        FK_medical_facility=randint(1, MEDICAL_FACILITIES_LIMIT)
    )
    session.add(vehicle)
    session.commit()

def generate_transfers(session):
    possible_drivers = get_users_with_role(session, 3)
    for i in range(0, TRANSFERS_LIMIT):
        transfer = Transfers(
            Start_time=fake.time(),
            Start_date=fake.date_between(start_date=START_DATE,
end_date='+1y'),
            FK_user=choice(possible_drivers),
            FK_vehicle=randint(1, VEHICLES_LIMIT),
            FK_facility_to=randint(1, MEDICAL_FACILITIES_LIMIT),
            FK_facility_from=randint(1, MEDICAL_FACILITIES_LIMIT)

```



```

    )
    session.add(transfer)
    session.commit()

def generate_patients(session):
    for i in range(0, PATIENTS_LIMIT):
        patient = Patients(
            Name=fake.first_name(),
            Surname=fake.last_name()
        )
        session.add(patient)
        session.commit()

def generate_admissions(session):
    for i in range(0, ADMISSIONS_LIMIT):
        admission_date = fake.date_between(START_DATE, END_DATE)
        discharge_date = None
        if fake.pybool():
            discharge_date = fake.date_between(admission_date, END_DATE)
        admission = Admissions(
            Admission_date=admission_date,
            Discharge_date=discharge_date,
            FK_hospital_ward=randint(1, HOSPITAL_WARDS_LIMIT),
            FK_patient=randint(1, PATIENTS_LIMIT)
        )
        session.add(admission)
        session.commit()

def generate_referrals(session):
    possible_doctors = get_users_with_role(session, 4)
    for i in range(0, REFERRALS_LIMIT):
        body_parts = None
        if fake.pybool():
            body_parts = choice(BODY_PARTS)
        referrals = Referrals(
            Referral_date=fake.date_between(START_DATE, END_DATE),
            Body_part=body_parts,
            FK_user=choice(possible_doctors),
            FK_patient=randint(1, PATIENTS_LIMIT)
        )
        session.add(referrals)
        session.commit()

def generate_medical_exams(session):
    for i in range(0, MEDICAL_EXAMS_LIMIT):
        medical_exam = Medical_exams(
            Name=fake.pystr(5, 45),
        )
        session.add(medical_exam)
        session.commit()

```

```

def generate_reservations(session):
    for i in range(0, RESERVATIONS_LIMIT):
        reservation = Reservations(
            Start_date=fake.date_between(start_date=START_DATE,
end_date='+1y'),
            Start_time=fake.time(),
            FK_patient=randint(1, PATIENTS_LIMIT),
            FK_medical_exam=randint(1, MEDICAL_EXAMS_LIMIT)
        )
        session.add(reservation)
        session.commit()

def generate_exam_offers(session):
    for i in range(0, RESERVATIONS_LIMIT):
        if fake.pybool():
            daily_limit = randint(1, DAILY_LIMIT_LIMIT)
        else:
            daily_limit = None
        exam_offers = Exam_offers(
            FK_medical_facility=randint(1, MEDICAL_FACILITIES_LIMIT),
            FK_medical_exam=randint(1, MEDICAL_EXAMS_LIMIT),
            Daily_limit=daily_limit
        )
        session.add(exam_offers)
        session.commit()

def generate_employments(session):
    possible_doctors = get_users_with_role(session, 4)
    for i in range(0, MEDICAL_EXAMS_LIMIT):
        employment_date = fake.date_between(EMPLOYMENT_DATE, END_DATE)
        dismissal_date = None
        if fake.pybool():
            dismissal_date = fake.date_between(employment_date, END_DATE)
        employment = Employments(
            FK_medical_facility=randint(1, MEDICAL_FACILITIES_LIMIT),
            FK_user=choice(possible_doctors),
            Employment_date=employment_date,
            Dismissal_date=dismissal_date
        )
        session.add(employment)
        session.commit()

def generate_passengers(session):
    for i in range(0, PASSENGERS_LIMIT):
        passenger = Passengers(
            FK_patient=randint(1, PATIENTS_LIMIT),
            FK_transfer=randint(1, TRANSFERS_LIMIT),
            Needs_care=NEEDS_CARE[randint(0, 1)],

```

```

        Status=choice(STATUS),
    )
    session.add(passenger)
    session.commit()

def generate_referrals_medical_exams(session):
    for i in range(0, REFERRALS_MEDICAL_EXAMS):
        referrals_medical_exams = Referrals_Medical_exams(
            FK_medical_exam=randint(1, MEDICAL_EXAMS_LIMIT),
            FK_referral=randint(1, REFERRALS_LIMIT),
        )
        session.add(referrals_medical_exams)
    session.commit()

def generate_users_transfers(session):
    possible_medics = get_users_with_role(session, 5)
    facility_from = randint(1, MEDICAL_FACILITIES_LIMIT)
    facility_to = randint(1, MEDICAL_FACILITIES_LIMIT)
    while facility_to == facility_from:
        facility_to = randint(1, MEDICAL_FACILITIES_LIMIT)
    for i in range(0, USERS_TRANSFERS_LIMIT):
        users_transfers = Users_Transfers(
            FK_user=choice(possible_medics),
            FK_transfer=randint(1, TRANSFERS_LIMIT)
        )
        session.add(users_transfers)
    session.commit()

def generate_all_silent(session):
    generate_users(session)
    generate_roles(session)
    generate_locations(session)
    generate_medical_facilities(session)
    generate_hospital_wards(session)
    generate_vehicles(session)
    generate_transfers(session)
    generate_patients(session)
    generate_admissions(session)
    generate_referrals(session)
    generate_medical_exams(session)
    generate_reservations(session)
    generate_exam_offers(session)
    generate_employments(session)
    generate_passengers(session)
    generate_referrals_medical_exams(session)
    generate_users_transfers(session)

def generate_all_verbose(session):
    generate_users(session)

```

```

    print("Generating Users finished")
    generate_roles(session)
    print("Generating Roles finished")
    generate_locations(session)
    print("Generating Locations finished")
generate_medical_facilities(session)
    print("Generating Medical_facilities finished")
    generate_hospital_wards(session)
    print("Generating Hospital_wards finished")
    generate_vehicles(session)
    print("Generating Vehicles finished")
    generate_transfers(session)
    print("Generating Transfers finished")
    generate_patients(session)
    print("Generating Patients finished")
    generate_admissions(session)
    print("Generating Admissions finished")
    generate_referrals(session)
    print("Generating Referrals finished")
    generate_medical_exams(session)
    print("Generating Medical_exams finished")
    generate_reservations(session)
    print("Generating Reservations finished")
    generate_exam_offers(session)
    print("Generating Exam_offers finished")
    generate_employments(session)
    print("Generating Employments finished")
    generate_passengers(session)
    print("Generating Passengers finished")
generate_referrals_medical_exams(session)
    print("Generating Referrals_Medical_exams finished")
    generate_users_transfers(session)
    print("Generating Users_Transfers finished")
    print()
    print("-----Database generated :)")

```

```

if __name__ == '__main__':
    Session = sessionmaker(bind=mysql_engine, autoflush=False)
    current_session = Session()
    # generate_all_verbose(current_session)
    generate_all_silent(current_session)
    current_session.close()

```

## Poprawki do etapu 4.

Skrócono metody pozyskujące dane z bazy danych. Metody wykorzystywane są do zapewnienia unikatowości danych zostały skrócone do jednej linijki kodu.

```
def get_logins(session):  
    return [user.Login for user in session.query(Users).all()]  
  
def get_supervisors(session):  
    return [medical_facility.FK_user for medical_facility in session.query(Medical_facilities).all()]  
  
def get_license_plates(session):  
    return [vehicle.License_plate for vehicle in session.query(Vehicles).all()]
```

Nie wykorzystano zmiennej globalnej session - zamiast tego session tworzona jest w mainie i przekazywana do metod jako argument.

Dodano zamknięcie sesji, żeby nie musieć polegać na garbage collectorze.

```
if __name__ == '__main__':  
    Session = sessionmaker(bind=mysql_engine)  
    current_session = Session()  
    generate_all_verbose(current_session)  
    # generate_all_silent(current_session)  
    current_session.close()
```

Dodano do tabeli Transfers dwa atrybuty - z której placówki wyjeżdża transport i do której placówki jedzie transport.

```
class Transfers(Base):  
    __tablename__ = 'Transfers'  
  
    Id_transfer = Column(Integer, primary_key=True, autoincrement=True)  
    Start_time = Column(Time)  
    Start_date = Column(Date)  
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))  
    FK_vehicle = Column(Integer, ForeignKey('Vehicles.Id_vehicle'))  
    FK_facility_from = Column(Integer, ForeignKey('Medical_facilities.Id_medical_facility'))  
    FK_facility_to = Column(Integer, ForeignKey('Medical_facilities.Id_medical_facility'))  
  
    passengers = relationship('Passengers', backref='transfers')  
    users_transfers = relationship('Users_Transfers', backref='transfers')
```

Zmieniono również losowanie nazw oddziałów jako losowych znaków. Losowane są nazwy z tablicy przechowującej możliwe nazwy oddziałów. Jedna placówka nie może mieć dwóch oddziałów o tej samej nazwie.

Ta zmiana zostanie wykorzystana w następnym etapie.



## Etap 5.

### Pomysły na interfejsy

1. Wyszukiwanie użytkowników (kierowców i opiekunów medycznych) oraz pacjentów jadących danym pojazdem w danym czasie:

The screenshot shows the 'Passengers' search interface. On the left is a dark sidebar with the 'MEDI-TRANS' logo and a navigation menu including Home, Statistics, Referrals, Employees, Transfers, Exams, Wards, Vehicles (highlighted), Help Center, and Settings. The main content area has a search bar at the top with the placeholder 'Search transfers, patients or help'. Below it, the title 'Passengers' is displayed. The central form is titled 'Find all passengers in vehicle' and contains a 'License plate' input field. Below this is a date picker for 'January 2018' with the 2nd day selected. To the right of the date picker are three checkboxes: 'Drivers' (unchecked), 'Medical care' (checked), and 'Patients' (unchecked). A blue 'Search' button is at the bottom of the form.

2. Lista placówek oferujących dane badanie w danym mieście:

The screenshot shows the 'Exam offers' search interface. The sidebar is identical to the previous one. The main content area has a search bar with the placeholder 'Search transfers, patients or help'. Below it, the title 'Exam offers' is displayed. The central form is titled 'Find exam offers in your city' and contains two input fields: one for the examination type (with a medical icon and 'X-ray' entered) and one for the location (with a city icon and 'Wroclaw' entered). A blue 'Search' button is to the right of the location field. Below the form is a table titled 'Exam offers' with three columns: 'LOCATION', 'MEDICAL FACILITY', and 'DAILY LIMIT'. The table lists six entries with their respective locations, medical facilities, and daily limits.

LOCATION	MEDICAL FACILITY	DAILY LIMIT
Curie-Skłodowskiej 58, 50-369 W...	Uniwersytecki Szpital Kliniczny	22
Kamieńskiego 73A, 51-124 Wrocl...	Wojewódzki Szpital Specjalistyczny we Wroc...	20
Tytusa Chałubińskiego 2a, 50-3...	Samodzielny Publiczny Szpital Nr 1 we Wrocław...	17
Poświęcka 8, 51-128 Wrocław	Uniwersytecki Szpital Kliniczny im. Jana Mikulic...	15
Olbińska 32, 50-233 Wrocław	SP ZOZ MSWiA we Wrocławiu	15
Koszarowa 5, 51-149 Wrocław	Wojewódzki Szpital Specjalistyczny im. J. Gro...	11

3. Liczba i lista pacjentów przebywających aktualnie w poszczególnych oddziałach danej placówki:

**MEDI-TRANS** Search transfers, patients or help John Doe

Home Statistics Referrals Employees Transfers Exams **Wards** Vehicles Help Center Settings

### Patients

Find all patients in Medical Facility

Uniwersytecki Szpital Medyczny w Chorzowie Search

HOSPITAL WARD	PATIENTS	DETAILS
Oddział Kardiologii	3746	→
<b>Oddział Kardiologiczny</b>	<b>8126</b>	<b>→</b>
Oddział Chirurgii Naczyniowej	8836	→
Oddział Laryngologiczny	1173	→
Oddział Anestezjologii	2739	→
Oddział Chirurgii	1762	→
Oddział Psychiatryczny	3746	→
Oddział Udarowy	3746	→

ID	NAME	SURNAME
12467	Zygmunt	Nazwisko
14789	Andrzej	Waligóra
19364	Anna	Fajna
12489	Magdalena	Śmieszna
32000	Zofia	Zofia
36782	Celina	Malina
00123	Jędrzej	Górka
00007	Joanna	Zygmunt

4. Lista placówek w danym mieście z informacją dotyczącą liczby wolnych miejsc na poszczególnych oddziałach:

**MEDI-TRANS** Search transfers, patients or help John Doe

Home Statistics Referrals Employees Transfers Exams **Wards** Vehicles Help Center Settings

### Places in wards

Find free places in your city

Wrocław Search

MEDICAL FACILITY	LOCATION	DETAILS
Uniwersytecki Szpital Kliniczny we Wroc...	Curie-Skłodowskiej 2...	→
<b>Wojewódzki Szpital Kliniczny we Wrocław...</b>	<b>Tytusa Chałubińskiego...</b>	<b>→</b>
Samodzielny Publiczny Szpital Nr 1 we...	Kamieńskiego 73A, 51...	→
Uniwersytecki Szpital Kliniczny im. Ja...	Koszarowa 5, 51-149 W...	→
SP ZOZ MSWiA we Wrocławiu	Olbiańska 32, 50-233 W...	→
Wojewódzki Szpital Specjalistyczny w...	Poświęcka 8, 51-128 W...	→
Dolnośląskie Centrum Chorób Płuc...	Grabiszyńska 105, 53-4...	→
Klinika Chirurgii i Urologii Dziecięcej	Curie-Skłodowskiej 5...	→

HOSPITAL WARDS	PLACES	FREE PLACES
Oddział Laryngologiczny	20	10
Oddział Chirurgii	30	12
Oddział Anestezjologii	32	19
Oddział Kardiologiczny	29	2
Oddział Intensywnej Terapii	45	11
Oddział Psychiatryczny	18	0



## 5. Lista skierowań wypisanych na dane badanie w konkretnym okresie:

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Referrals - Overview

X-ray

Pick dates

< July 2021 >

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Search

Referrals for specific exam

Referrals

PATIENT	DOCTOR	DATE	BODY PART
Jan Brzechwa	August Oetker	19.07.2021	Prawa noga
Anna Zaradna	Jan Dzbani	20.07.2021	Szczeka
Tomasz Problem	Gajusz Pompejusz	20.07.2021	Lewa raka
Jan Kowalski	Sebastian Bach	21.07.2021	
Narczy Baran	Kamil Słimak	23.07.2021	
Hanna Wanna	Lucyna Fachowa	25.07.2021	Głowa
Wiktor Gołab	Zenon Mały	25.07.2021	Brzuch
Roman Zupa	Juliusz Cezar	25.07.2021	
Hiacynt Fioletowy	Anna Mammamia	25.07.2021	Barki

## 6. Lista pacjentów najczęściej korzystających z przejazdów z informacją dotyczącą liczby przejazdów, na których pasażerem był pacjent:

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

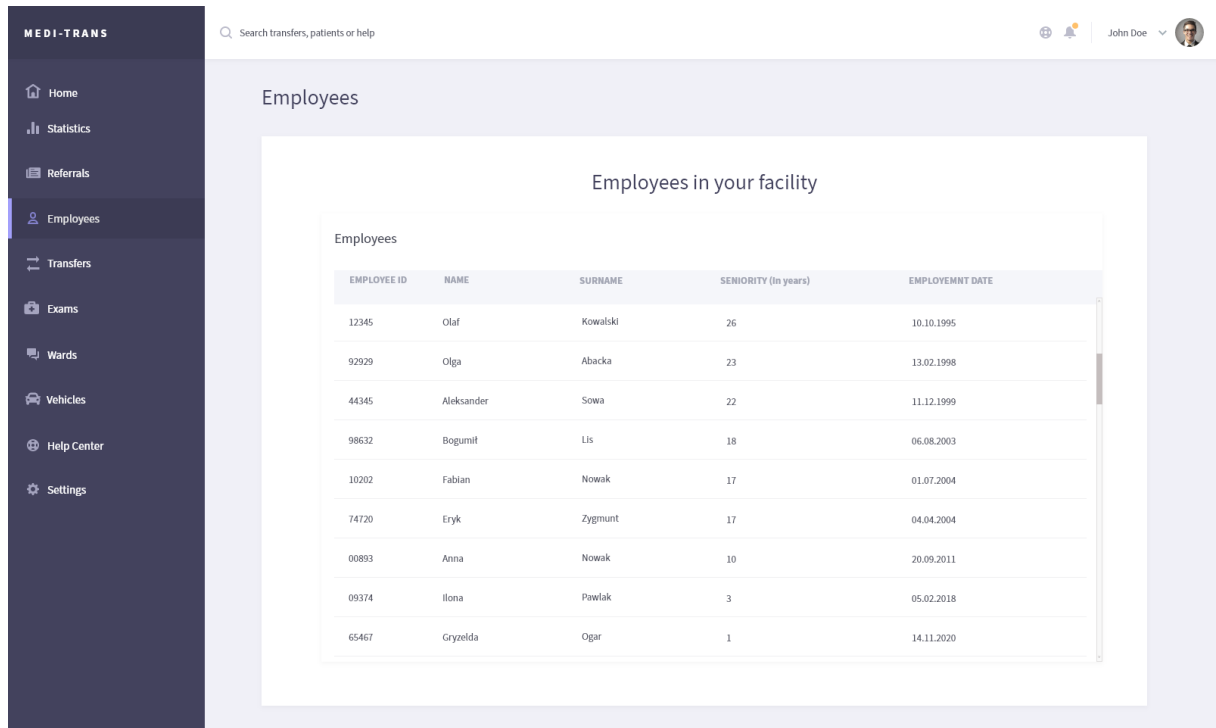
John Doe

Most active patients

Patients

ID	NAME	SURNAME	TRANSPORT AMOUNT
00321	Zbigniew	Stonoga	145
12345	Anna	Fajrant	144
67890	Tomasz	Qwerty	128
72845	Jan	Opolski	76
00453	Narczy	Chwalebny	33
32048	Hanna	Joanna	33
29204	Wiktor	Zółty	33
11111	Roman	Chwalipieta	29
66666	Hiacynt	Bazodanowy	17
00462	Zuzanna	Surowy	17

7. Lista pracowników zatrudnionych w danej placówce z informacją o stażu pracy każdego pracownika:

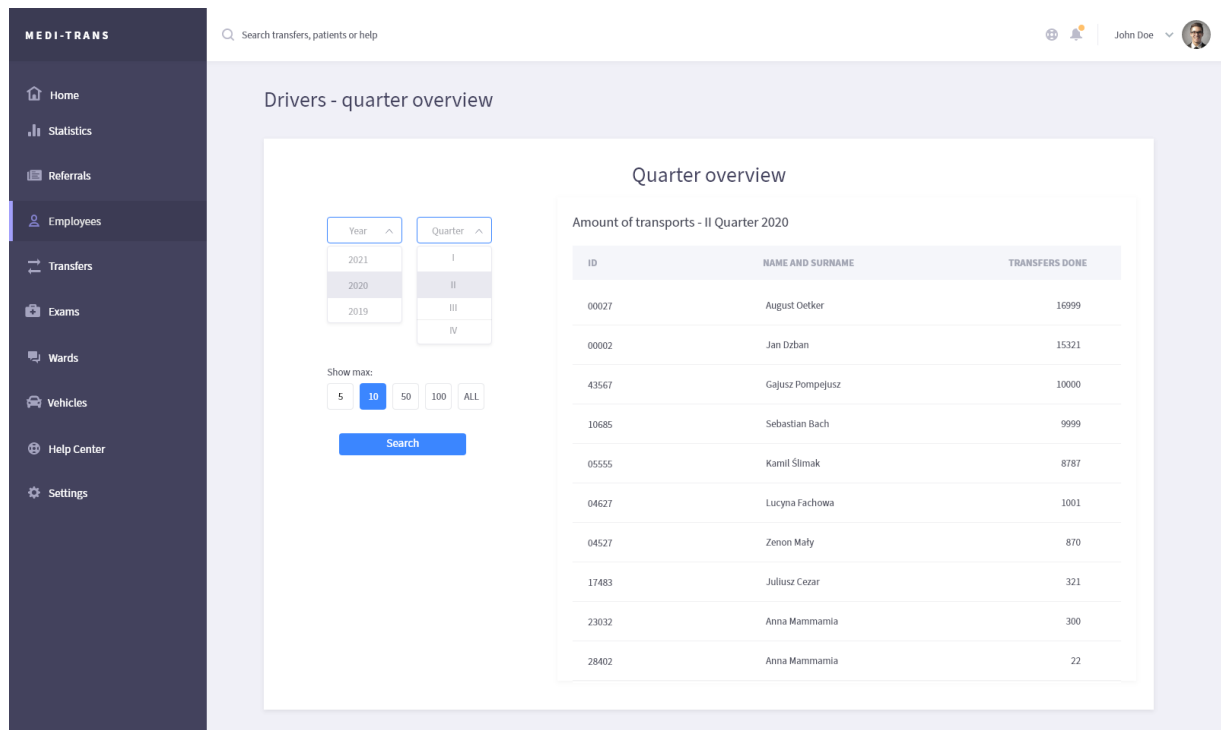


Employees

Employees in your facility

EMPLOYEE ID	NAME	SURNAME	SENIORITY (in years)	EMPLOYMENT DATE
12345	Olaf	Kowalski	26	10.10.1995
92929	Olga	Abacka	23	13.02.1998
44345	Aleksander	Sowa	22	11.12.1999
98632	Bogumił	Lis	18	06.08.2003
10202	Fabian	Nowak	17	01.07.2004
74720	Eryk	Zygmunt	17	04.04.2004
00893	Anna	Nowak	10	20.09.2011
09374	Ilona	Pawlak	3	05.02.2018
65467	Gryzelda	Ogar	1	14.11.2020

8. Lista kierowców wykonujących w danym kwartale najwięcej kursów:



Drivers - quarter overview

Quarter overview

Amount of transports - II Quarter 2020

ID	NAME AND SURNAME	TRANSFERS DONE
00027	August Oetker	16999
00002	Jan Dżban	15321
43567	Gajusz Pompejusz	10000
10685	Sebastian Bach	9999
05555	Kamil Ślimak	8787
04627	Lucyna Fachowa	1001
04527	Zenon Mały	870
17483	Juliusz Cezar	321
23032	Anna Mammamia	300
28402	Anna Mammamia	22

## 9. Lista pojazdów znajdujących się w danej placówce:

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Vehicles in Medical Facility

Vehicles

LICENSE PLATE	BRAND	MODEL	SEATS
ESI 28D3	Audi	Q8	5
ESI 34FJ	BMW	M3	4
ESI 733K	Skoda	Octavia	4
ESI H345	Seat	Ibiza	8
ESI JK33	Fiat	126p	20
ESI R5R5	Fiat	Punto	18
ESI 9909	Chevrolet	Camaro	17
ESI 44J2	Honda	Civic	9
ESI 98XL	Honda	CRV	7
ESI 7811	KIA	Rio	17

License plate...

Search

Number of seats:

1

4

5

6

7

8

9

10

13

17

18

19

20

24

ALL

Brand...

Search

Model...

Search

## 10. Lista placówek z informacją dotyczącą liczby wykonanych z nich przejazdów w danym okresie:

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Transfers - Overview

Your city...

Pick dates

<

July 2021

>

Su	Mo	Tu	We	Th	Fr	Sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Search

Transfers by Medical Facility

Amount of transports commissioned by Medical Facilities - 18.07.2021 - 26.07.2021

ID	MEDICAL FACILITY	EXECUTED TRANSFERS
00027	Uniwersytecki Centrum Kliniczne w Gdańsku	16999
00002	Wojskowy Instytut Medyczny w Warszawie	15321
43567	Centralny Szpital Kliniczny Mniszterstwa Spraw Wewnętrznych w Warszawie	10000
10685	Krakowski Szpital Specjalistyczny im. Jana Pawła II	9999
05555	Instytut Centrum Zdrowia Matki Polki w Łodzi	8787
04627	Uniwersytecki Szpital Dziecięcy w Krakowie	1001
04527	Śląskie Centrum Chorób Serca w Zabrzu	870
17483	Szpital Uniwersytecki Nr 1 im. Dr. A. Jurasza w Bydgoszczy	803
23032	Uniwersytecki Szpital Kliniczny w Białymstoku	799
28402	Świętokrzyskie Centrum Onkologii w Kielcach	90

11. Lista oddziałów danego typu znajdujących się w danym mieście z informacją o placówce, w której znajduje się oddział i liczbie miejsc na oddziale:

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Wards in city

Find specific Medical Ward in your city

Oddział Kardiologii

Wroclaw

Search

Oddział Kardiologii in Wroclaw

LOCATION	MEDICAL FACILITY	MEDICAL FACILITY	PLACES
Curie-Skłodowskiej 58, 50-369 W...	Uniwersytecki Szpital Kliniczny	Oddział Kardiologii	560
Kamieńskiego 73A, 51-124 Wroc...	Wojewódzki Szpital Specjalistyczny we Wroc...	Oddział Kardiologii	450
Tytusa Chałubińskiego 2a, 50-3...	Samodzielny Publiczny Szpital Nr 1 we Wrocław...	Oddział Kardiologii	320
Poświęcka 8, 51-128 Wrocław	Uniwersytecki Szpital Kliniczny im. Jana Mikulic...	Oddział Kardiologii	170
Olbiańska 32, 50-233 Wrocław	SP ZOZ MSWiA we Wrocławiu	Oddział Kardiologii	166
Koszarowa 5, 51-149 Wrocław	Wojewódzki Szpital Specjalistyczny im. J. Gro...	Oddział Kardiologii	79

12. Lista użytkowników (kierowców i opiekunów medycznych) uczestniczących w przejazdach odbywających się w danym miesiącu:

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Transfers - overview

Uniwersytecki Szpital Kliniczny

Curie-Skłodowskiej 58  
50-369 Wrocław

Summary

Print

Summary to

Uniwersytecki Szpital Kliniczny  
Curie-Skłodowskiej 58  
50-369 Wrocław

Start term: March 1, 2020  
End term: March 31, 2020

#	NAME	SURNAME	ROLE	TRANSFER AMOUNT
1	Wiktor	Faradajski	Driver	52
2	Anna	Zaradna	Care	47
3	Józef	Piłsudski	Driver	31
4	Józefina	Piłsudska	Care	31
5	Tomasz	Hajto	Driver	29

Total drivers: 258  
Total cares: 177

Total all: 455

Submit

### 13. Tworzenie transportu:

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Transfers menu

Scheduled transfers

Book transfer

Modify transfer

Add patient

Start facility

Destination facility

Transport date

Search

START FACILITY	DESTINATION FACILITY	TRANSFER DATE	
SP ZOZ MSWIA we Wrocławiu	Uniwersytecki Szpital Kliniczny	20.11.2021	→
Wojewódzki Szpital Specjalistyczny Im. J. Gro...	Wojewódzki Szpital Specjalistyczny we Wroc...	21.12.2021	→
Uniwersytecki Szpital Kliniczny	Samodzielny Publiczny Szpital Nr 1 we Wrocław...	01.12.2021	→
Wojewódzki Szpital Specjalistyczny we Wroc...	Uniwersytecki Szpital Kliniczny Im. Jana Mikulic...	30.11.2021	→
Wojewódzki Szpital Specjalistyczny we Wroc...	SP ZOZ MSWIA we Wrocławiu	24.11.2021	→
SP ZOZ MSWIA we Wrocławiu	Wojewódzki Szpital Specjalistyczny Im. J. Gro...	01.01.2022	→
Samodzielny Publiczny Szpital Nr 1 we Wrocław...	Uniwersytecki Szpital Kliniczny	03.12.2021	→
Wojewódzki Szpital Specjalistyczny Im. J. Gro...	SP ZOZ MSWIA we Wrocławiu	10.12.2021	→
Samodzielny Publiczny Szpital Nr 1 we Wrocław...	Wojewódzki Szpital Specjalistyczny we Wroc...	27.11.2021	→

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Book transfer

Choose destination

City

Find facility...

Search

Medical Facilities

MEDICAL FACILITY	LOCATION	WARDS
Uniwersytecki Szpital Kliniczny we Wrocławiu	Curie-Skłodowskiej 2, 50-110 Wrocław	→
Wojewódzki Szpital Kliniczny we Wrocławiu	Tytusa Chałubińskiego 13, 53-000 Wrocław	→
Samodzielny Publiczny Szpital Nr 1 we Wrocławiu	Kamieńskiego 73A, 51-120 Wrocław	→
Uniwersytecki Szpital Kliniczny	Koszarowa 5, 51-149 Wrocław	→
SP ZOZ MSWIA we Wrocławiu	Ołbińska 32, 50-233 Wrocław	→
Wojewódzki Szpital Specjalistyczny we Wrocławiu	Poświęcka 8, 51-128 Wrocław	→

Wojewódzki Szpital Specjalistyczny we Wrocławiu

HOSPITAL WARDS	PLACES	FREE PLACES
Oddział Laryngologiczny	20	10
Oddział Chirurgii	30	12
Oddział Anestezjologii	32	19
Oddział Kardiologiczny	29	2
Oddział Intensywnej Terapii	45	11

Next

1

2

3

4

Step 1

Step 2

Step 3

Step 4

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Book transfer

Choose date of transport

2 January 2018

<January 2018>

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

21

22

21

20

19

18

37

38

37

36

35

34

Next

Step 1

Step 2

Step 3

Step 4

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Book transfer

Choose vehicle and driver

Available drivers

ID	NAME	SURNAME
00321	Zbigniew	Stonoga
12345	Anna	Fajrant
67890	Tomasz	Qwerty
72845	Jan	Opolski
00453	Narczyz	Chwalebny
32048	Hanna	Joanna
29204	Wiktor	Zółty
11111	Roman	Chwalipełta

Vehicles

LICENSE PLATE	BRAND	MODEL	SEATS
ESI 28D3	Audi	Q8	5
ESI 34FJ	BMW	M3	4
ESI 733K	Skoda	Octavia	4
ESI H345	Seat	Ibiza	8
ESI JK33	Fiat	126p	20
ESI R5R5	Fiat	Punto	18
ESI 9909	Chevrolet	Camaro	17
ESI 44J2	Honda	Civic	9

Next

Step 1

Step 2

Step 3

Step 4

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Book transfer

Choose patients

Surname

Name

ID

Search

Available medics

ID	NAME	SURNAME	STATUS	NEEDS CARE
00221	Zbigniew	Stonoga	Lying	YES
12345	Anna	Fajrant	Sitting	NO
67890	Tomasz	Qwerty	Walking	NO
72845	Jan	Opolski	Walking	YES
00453	Narczyz	Chwalebny	Lying	YES
32048	Hanna	Joanna	Walking	NO

Choose medical care

Medic required

Available medics

ID	NAME	SURNAME
00321	Zbigniew	Stonoga
12345	Anna	Fajrant
67890	Tomasz	Qwerty
72845	Jan	Opolski
00453	Narczyz	Chwalebny
32048	Hanna	Joanna

Next

Step 1

Step 2

Step 3

Step 4

MEDI-TRANS

Home

Statistics

Referrals

Employees

Transfers

Exams

Wards

Vehicles

Help Center

Settings

Search transfers, patients or help

John Doe

Book transfer

Summary

FROM

Wojewódzki Szpital Psychiatryczny  
ul. Sieradzka 3, 98-290 Warta

TO

Wojewódzki Szpital Kliniczny we Wrocławiu  
ul. Tytusa Chałubińskiego 13, 53-000 Wrocław  
Oddział Anestezjologii

WHEN

2 January 2018  
21:37

DRIVER

Zbigniew Stonoga

VEHICLE

Fiat 126p  
ESI JK33

MEDICAL CARE

Anna Fajrant

PATIENTS

1.

Tomasz Qwerty  
Status: Walking  
Needs care: NO

2.

Narczyz Chwalebny  
Status: Lying  
Needs care: YES

Book transfer

Edit

Step 1

Step 2

Step 3

Step 4

## Raporty i funkcje wyszukiwania

Funkcje agregujące służą do podsumowania danych w bazie. Należą do nich m.in. użyte przez nas:

- zliczanie - COUNT() (Query 4, 5, 7, 8)
- obliczanie średniej - AVG() (Query 7)
- wyznaczanie maksimum - MAX() (Query 9)

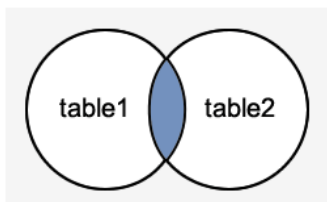
Wyszukiwanie (filtrowanie) danych w bazie odbywa się przy pomocy komendy WHERE.

ORDER BY używane jest do sortowania danych będących np. rezultatem SELECT. Możliwe jest nadanie kolejności malejącej (DESC) i rosnącej (ASC).

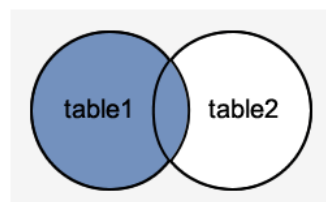
Operator UNION używany jest do łączenia dwóch lub więcej wyników komendy SELECT:

- Każdy SELECT wykorzystany w UNION musi mieć tę samą liczbę kolumn.
- Kolumny muszą mieć podobne typy danych.
- Kolumny w każdym SELECT muszą być w tej samej kolejności.

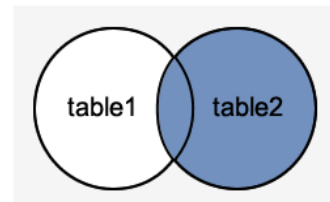
INNER JOIN:



LEFT JOIN:



RIGHT JOIN:



### Kwerendy SQL:

1. Wyszukiwanie użytkowników (kierowców i opiekunów medycznych) oraz pacjentów jadących danym pojazdem w danym czasie:

```
SELECT users.Name, Surname, 'Driver' As Rola
FROM transfers
left JOIN users ON users.Id_user = transfers.FK_user
left JOIN vehicles ON vehicles.Id_vehicle = transfers.FK_vehicle
WHERE transfers.Start_date BETWEEN '&userDate1' AND '&userDate2' AND
vehicles.License_plate = '&plate'
```

UNION

```
SELECT users.Name, Surname, 'Medic' As Rola
FROM medical_transport.users
INNER JOIN users_transfers ON users_transfers.FK_user = users.Id_user
INNER JOIN transfers ON transfers.Id_transfer = users_transfers.FK_transfer
INNER JOIN vehicles ON vehicles.Id_vehicle = transfers.FK_vehicle
WHERE transfers.Start_date BETWEEN '&userDate1' AND '&userDate2' AND
vehicles.License_plate = '&plate'
```

UNION

```
SELECT patients.Name, patients.Surname, 'Patient' As Rola
FROM medical_transport.patients
INNER JOIN passengers ON passengers.FK_patient = patients.Id_patient
INNER JOIN transfers ON transfers.Id_transfer = passengers.FK_transfer
INNER JOIN vehicles ON vehicles.Id_vehicle = transfers.FK_vehicle
WHERE transfers.Start_date BETWEEN '&userDate1' AND '&userDate2' AND
vehicles.License_plate = '&plate';
```



## 2. Lista placówek oferujących dane badanie w danym mieście:

```
SELECT medical_facilities.Name
FROM medical_transport.medical_facilities
INNER JOIN locations ON locations.Id_location =
medical_facilities.FK_location
INNER JOIN exam_offers ON exam_offers.FK_medical_facility =
medical_facilities.Id_medical_facility
INNER JOIN medical_exams ON medical_exams.Id_medical_exam =
exam_offers.FK_medical_exam
WHERE medical_exams.Name = '&exam_name' AND locations.City = '&city';
```

## 3. Liczba wolnych i zajętych miejsc na poszczególnych oddziałach w danej placówce:

```
SELECT medical_facilities.Name, hospital_wards.name, hospital_wards.places,
hospital_wards.places - COUNT(admissions.Id_admission) AS free_places
FROM admissions
INNER JOIN hospital_wards ON hospital_wards.Id_hospital_ward =
admissions.FK_hospital_ward
INNER JOIN medical_facilities ON medical_facilities.Id_medical_facility =
hospital_wards.FK_medical_facility
INNER JOIN locations ON locations.Id_location =
medical_facilities.FK_location
WHERE isnull(admissions.Discharge_date) AND locations.city = '&city'
GROUP BY hospital_wards.Id_hospital_ward;
```

## 4. Liczba skierowań wypisanych na dane badanie w konkretnym okresie:

```
SELECT COUNT(referrals.Id_referral)
FROM referrals
INNER JOIN refferals_medical_exams ON refferals_medical_exams.FK_referral =
referrals.Id_referral
INNER JOIN medical_exams ON medical_exams.Id_medical_exam =
refferals_medical_exams.FK_medical_exam
WHERE medical_exams.Name = '&exam_name' AND referrals.Referral_date BETWEEN
'&date_start' AND '&date_end';
```

## 5. Lista pacjentów najczęściej korzystających z przejazdów (dziesięciu pacjentów):

```
SELECT patients.name, patients.surname, COUNT(*) AS transports
FROM patients
LEFT JOIN passengers ON passengers.FK_patient = patients.Id_patient
GROUP BY patients.Id_patient
ORDER BY transports DESC
LIMIT 10;
```

6. Lista pracowników zatrudnionych w danej placówce z informacją o stażu pracy każdego pracownika:

```
SELECT users.Name, users.Surname, employments.Employment_date,  
floor(datediff(now(), employments.Employment_date)/365) AS seniority  
FROM users  
INNER JOIN employments ON employments.FK_user = users.Id_user  
INNER JOIN medical_facilities ON medical_facilities.Id_medical_facility =  
employments.FK_medical_facility  
WHERE Dismissal_date IS NULL AND medical_facilities.Name = '&name'  
ORDER BY seniority DESC;
```

7. Średnia liczba przejazdów kierowców:

```
SELECT AVG(transfers_number) AS Average_Transfers_For_Driver FROM (  
SELECT users.Name, users.Surname, COUNT(*) AS transfers_number  
FROM users  
INNER JOIN roles ON roles.FK_user = users.Id_user  
INNER JOIN transfers ON transfers.FK_user = users.Id_user  
GROUP BY users.Id_user  
ORDER BY transfers_number DESC) AS T;
```

8. Lista placówek z informacją dotyczącą liczby wykonanych z nich przejazdów w danym okresie:

```
SELECT medical_facilities.Name, COUNT(transfers.Id_transfer) AS Transfers  
from medical_facilities  
INNER JOIN transfers ON transfers.FK_facility_to =  
medical_facilities.Id_medical_facility  
GROUP BY medical_facilities.Id_medical_facility  
ORDER BY Transfers DESC  
LIMIT 100;
```

9. Placówka z największym dziennym limitem badań na każde badanie:

```
SELECT medical_exams.Name as Exam, MAX(exam_offers.Daily_limit) AS 'Max  
Limit' , medical_facilities.Name AS Facility  
FROM exam_offers  
LEFT JOIN medical_exams ON medical_exams.Id_medical_exam =  
exam_offers.FK_medical_exam  
LEFT JOIN medical_facilities ON medical_facilities.Id_medical_facility =  
exam_offers.FK_medical_facility  
GROUP BY medical_exams.Name;
```

## 10. Lista różnych oddziałów w danym mieście:

```
SELECT DISTINCT hospital_wards.Name
FROM hospital_wards
LEFT JOIN medical_facilities ON medical_facilities.Id_medical_facility =
hospital_wards.FK_medical_facility
LEFT JOIN locations ON locations.Id_location = medical_facilities.FK_location
WHERE locations.City = '&city';
```

## Skrypt testujący napisane kwerendy

```
def execute_query_1(session):
    result = session.execute(
        'SELECT users.Name, Surname, "Driver" As Rola FROM transfers left JOIN
users ON users.Id_user = transfers.FK_user '
        'left JOIN vehicles ON vehicles.Id_vehicle = transfers.FK_vehicle WHERE
transfers.Start_date '
        'BETWEEN :userDate1 AND :userDate2 AND vehicles.License_plate = :plate
UNION SELECT users.Name, Surname, '
        '"Medic" As Rola FROM medical_transport.users INNER JOIN
users_transfers ON users_transfers.FK_user = users.Id_user '
        'INNER JOIN transfers ON transfers.Id_transfer =
users_transfers.FK_transfer INNER JOIN vehicles '
        'ON vehicles.Id_vehicle = transfers.FK_vehicle WHERE
transfers.Start_date BETWEEN :userDate1 AND :userDate2 '
        'AND vehicles.License_plate = :plate UNION SELECT patients.Name,
patients.Surname, "Patient" As Rola '
        'FROM medical_transport.patients INNER JOIN passengers ON
passengers.FK_patient = patients.Id_patient '
        'INNER JOIN transfers ON transfers.Id_transfer = passengers.FK_transfer
INNER JOIN vehicles '
        'ON vehicles.Id_vehicle = transfers.FK_vehicle WHERE
transfers.Start_date BETWEEN :userDate1 AND :userDate2 '
        'AND vehicles.License_plate = :plate',
        {"userDate1": "2021-08-09", "userDate2": "2022-01-01", "plate": "8EE
G59"})
```

```
print('\nQuery 1:')
for r in result:
    print(r)
```

```
def execute_query_2(session):
    result = session.execute(
        'SELECT medical_facilities.Name FROM
medical_transport.medical_facilities INNER JOIN locations ON '
        'locations.Id_location = medical_facilities.FK_location INNER JOIN
exam_offers ON '
        'exam_offers.FK_medical_facility =
medical_facilities.Id_medical_facility INNER JOIN medical_exams ON '
        'medical_exams.Id_medical_exam = exam_offers.FK_medical_exam WHERE
medical_exams.Name = :exam_name AND locations.City = :city'
```

```

, {"exam_name": "OsaZyqqKWgizkHvNxMIqMpFdnQi", "city": "Eatonton"})

print('\nQuery 2: ')
for r in result:
    print(r)

def execute_query_3(session):
    result = session.execute(
        'SELECT medical_facilities.Name, hospital_wards.name,
hospital_wards.places, hospital_wards.places - COUNT(admissions.Id_admission)
AS free_places '
        'FROM admissions INNER JOIN hospital_wards ON
hospital_wards.Id_hospital_ward = admissions.FK_hospital_ward '
        'INNER JOIN medical_facilities ON
medical_facilities.Id_medical_facility = hospital_wards.FK_medical_facility '
        'INNER JOIN locations ON locations.Id_location =
medical_facilities.FK_location '
        'WHERE isnull(admissions.Discharge_date) AND locations.city = :city
GROUP BY hospital_wards.Id_hospital_ward'
        , {"city": "Gaymouth"})

    print('\nQuery 3')
    for r in result:
        print(r)

def execute_query_4(session):
    result = session.execute(
        'SELECT COUNT(referrals.Id_referral) FROM referrals INNER JOIN
refferals_medical_exams '
        'ON refferals_medical_exams.FK_referral = referrals.Id_referral INNER
JOIN medical_exams '
        'ON medical_exams.Id_medical_exam =
refferals_medical_exams.FK_medical_exam '
        'WHERE medical_exams.Name = :exam_name AND referrals.Referral_date
BETWEEN :date_start AND :date_end'
        , {"exam_name": "lSPPZuuXNxfvVGqVfdUVqrkGPdqUqsKc", "date_start":
"2021-08-01", "date_end": "2022-01-01"})

    print('\nQuery 4')
    for r in result:
        print(r)

def execute_query_5(session):
    result = session.execute(
        'SELECT patients.name, patients.surname, COUNT(*) AS transports FROM
patients LEFT JOIN passengers '
        'ON passengers.FK_patient = patients.Id_patient GROUP BY
patients.Id_patient ORDER BY transports DESC LIMIT 10')

    print('\nQuery 5')

```

```

        for r in result:
            print(r)

def execute_query_6(session):
    result = session.execute(
        'SELECT users.Name, users.Surname, employments.Employment_date,
        floor(datediff(now(), employments.Employment_date)/365) '
        'AS seniority FROM users INNER JOIN employments ON employments.FK_user
= users.Id_user '
        'INNER JOIN medical_facilities ON
medical_facilities.Id_medical_facility = employments.FK_medical_facility '
        'WHERE Dismissal_date IS NULL AND medical_facilities.Name = :name ORDER
BY seniority DESC'
        , {"name": "ZoonhFzqgYRmeqJHmxjY"})

    print('\nQuery 6')
    for r in result:
        print(r)

def execute_query_7(session):
    result = session.execute(
        'SELECT AVG(transfers_number) AS Average_Transfers_For_Driver FROM
(SELECT users.Name, users.Surname, COUNT(*) '
        'AS transfers_number FROM users INNER JOIN roles ON roles.FK_user =
users.Id_user INNER JOIN transfers '
        'ON transfers.FK_user = users.Id_user GROUP BY users.Id_user ORDER BY
transfers_number DESC) AS T')

    print('\nQuery 7')
    for r in result:
        print(r)

def execute_query_8(session):
    result = session.execute(
        'SELECT medical_facilities.Name, COUNT(transfers.Id_transfer) AS
Transfers from medical_facilities '
        'INNER JOIN transfers ON transfers.FK_facility_to =
medical_facilities.Id_medical_facility '
        'GROUP BY medical_facilities.Id_medical_facility ORDER BY Transfers
DESC LIMIT 10')

    print('\nQuery 8')
    for r in result:
        print(r)

def execute_query_9(session):
    result = session.execute(
        'SELECT medical_exams.Name as Exam, MAX(exam_offers.Daily_limit) AS
"Max Limit", medical_facilities.Name AS Facility '

```

```

        'FROM exam_offers LEFT JOIN medical_exams ON
medical_exams.Id_medical_exam = exam_offers.FK_medical_exam '
        'LEFT JOIN medical_facilities ON medical_facilities.Id_medical_facility
= exam_offers.FK_medical_facility '
        'GROUP BY medical_exams.Name LIMIT 10')

```

```

print('\nQuery 9')
for r in result:
    print(r)

```

```

def execute_query_10(session):
    result = session.execute(
        'SELECT DISTINCT hospital_wards.Name FROM hospital_wards LEFT JOIN
medical_facilities '
        'ON medical_facilities.Id_medical_facility =
hospital_wards.FK_medical_facility LEFT JOIN locations '
        'ON locations.Id_location = medical_facilities.FK_location WHERE
locations.City = :city'
        , {"city": "East Robertomouth"})

```

```

print('\nQuery 10')
for r in result:
    print(r)

```

```

def execute_queries():
    execute_query_1(current_session)
    execute_query_2(current_session)
    execute_query_3(current_session)
    execute_query_4(current_session)
    execute_query_5(current_session)
    execute_query_6(current_session)
    execute_query_7(current_session)
    execute_query_8(current_session)
    execute_query_9(current_session)
    execute_query_10(current_session)

```

```

if __name__ == '__main__':
    Session = sessionmaker(bind=mysql_engine, autoflush=False)
    current_session = Session()
    execute_queries()
    current_session.close()

```

## Poprawki do etapu 5.

Wykorzystano klauzulę HAVING.

W przeciwieństwie do klauzuli WHERE, która dotyczy pojedynczych wierszy tabeli, klauzula HAVING dotyczy grup wierszy. Klauzula HAVING wykorzystywana jest, gdy używane jest grupowanie lub funkcje agregujące.

```
SELECT users.Name, users.Surname, employments.Employment_date,  
floor(datediff(now(), employments.Employment_date)/365) AS seniority  
FROM users  
INNER JOIN employments ON employments.FK_user = users.Id_user  
INNER JOIN medical_facilities ON medical_facilities.Id_medical_facility =  
employments.FK_medical_facility  
WHERE Dismissal_date IS NULL AND medical_facilities.Name = '&name'  
HAVING seniority > 5  
ORDER BY seniority DESC;
```

## Etap 6.

### **Funkcja explain przed dodaniem indeksów**

Kolumny:

- **ID** – Kolumna id zawiera numer zapytania, którego dotyczy. W przypadku zapytań z podzapytaniami, podzapytania w dyrektywie FROM oraz zapytań z dyrektywą UNION podzapytania numerowane są (zazwyczaj) w kolejności ich występowania w zapytaniu. (Patrz. Query 1)  
Możemy zobaczyć, że MySQL zawsze kopiuje wyniki zapytania UNION do tabeli tymczasowej, a następnie z tabeli tymczasowej zwraca nam otrzymany rezultat. Stąd mamy 3, a nie 2 wiersze.
- **SELECT\_TYPE** – W kolumnie select\_type otrzymujemy informacje o tym jakiego typu jest dane zapytanie.
  - PRIMARY zostaje oznaczona najbardziej zewnętrzną składową w badanym zapytaniu;
  - SIMPLE otrzymujemy jeżeli nie mamy żadnych podzapytań w danym zapytaniu;
  - SUBQUERY to podzapytania, które występują w klauzuli;
  - SELECT zapytania pierwotnego;
  - DERIVED to podzapytania w klauzuli FROM zapytania pierwotnego;
  - Typ UNION otrzymują drugie i kolejne części zapytania UNION. UNION RESULT (jak już wcześniej wspomniałem) jest efektem pracy MySQL – zwraca rezultat z tymczasowej tabeli w przypadku zapytań z UNION.
- **TABLE** – kolumna table mówi o tym jakiej tabeli dotyczy dane zapytanie. Odczytując wartości z tej kolumny możemy zobaczyć jaką kolejność optymalizator zapytań MySQL zdecydował się zastosować do danego zapytania.
- **TYPE** – kolumna type pokazuje w jaki sposób MySQL musi przetworzyć wiersze w tabeli.
  - ALL – oznacza, że należy przeskanować wszystkie wiersze w tabeli,
  - INDEX – MySQL musi przeskanować wszystkie wiersze w tabeli, ale może wykonać to w porządku w jakim jest przechowywane w indeksie, co może zaoszczędzić czas wykonywania zapytania gdyż nie trzeba go już sortować (ale dalej musimy uzyskać dostęp do pełnej tabeli na dysku),
  - RANGE – MySQL może użyć indeksu do przeszukania tylko potrzebnych w danym zapytaniu wartości,
  - REF – MySQL może przeszukać jedynie indeks do znalezienia potrzebnych wartości. Może występować przypadek REF\_OR\_NULL gdzie potrzebny jest jeszcze dodatkowy dostęp do pobrania wartości odpowiadających NULL,
  - EQ\_REF – MySQL zdecydował do użycia indeksu głównego lub unikalnego do zwrócenia wartości. Jest to bardzo szybki dostęp, bo przy natrafieniu na wartość MySQL może ją zwrócić bez szukania dalszych, także dopasowanych wartości,



- **CONST** – zazwyczaj występuje w przypadku użycia w klauzuli WHERE wartości z indeksu głównego tabeli.
- **POSSIBLE\_KEYS** – ta kolumna mówi o tym, które indeksy mogą być użyte przy obliczaniu wyniku zapytania. Jest tworzona na początku procesu optymalizacji
- **KEY** – wskazuje, który indeks będzie użyty przez MySQL podczas wykonywania zapytania.
- **KEY\_LEN** – podaje informacje ile bajtów indeksu zostanie użytych przez MySQL podczas wykonywania zapytania.
- **REF** – Ta kolumna informuje o tym które kolumny z indeksów podany w kolumnie key zostaną użyte podczas wykonywania zapytania
- **ROWS** – podaje informacje ile średnio rekordów w bazie danych będzie należało przejrzeć żeby wydobyć dane niezbędne w zapytaniu. Jeżeli mamy do czynienia ze złożonym zapytaniem to informacja dotyczy pojedynczego wykonania pętli dla JOIN. Informacje te mogą być mocno nieprecyzyjne.
- **EXTRA** – Najważniejsze wartości jakie mogą pojawić się w tej kolumnie:
  - Using index – podczas wykonywania zapytania użyty będzie indeks zamiast odczytu bezpośredniego z tabeli
  - Using where – MySQL będzie musiał przefiltrować wyniki otrzymane z silnika bazy danych. Często jest to informacja, którą warto przemyśleć pod kątem stworzenia nowego (lub modyfikacji istniejących) indeksów.
  - Using temporary – MySQL użyje tabeli tymczasowej do sortowania wyników
  - Using filesort – wymagane będzie użycie sortowania po wyciągnięciu danych z bazy zamiast wyciągnięcia ich w wymaganym porządku

**Optymalizacja zapytań** polega na zapisywaniu kwerend w taki sposób, aby osiągnąć najniższą złożoność obliczeniową, tzn. aby kwerenda była wykonywana w bazie jak najszybciej. Dodanie indeksów do bazy jest sposobem jej optymalizacji, bo zmniejsza czas potrzebny na wyszukanie rekordów.

Dzięki optymalizacji można zmniejszyć czas oczekiwania na wynik co jest szczególnie widoczne przy bazach z dużą ilością danych i takich gdzie ilość danych szybko narasta.

Niemożliwe było usunięcie indeksów w całości – do wygenerowania bazy wymagane jest określenie dla tabel indeksów PRIMARY KEY. Wobec tego niemożliwe było sprawdzenie wyników funkcji explain w bazie danych w ogóle nie używającej indeksów. Baza danych musi mieć przynajmniej indeks PRIMARY KEY.

**Wyniki funkcji explain (pierwszy screen – wynik funkcji explain dla bazy bez indeksów, drugi screen – wynik funkcji explain dla bazy z indeksami):**

1. Wyszukiwanie użytkowników (kierowców i opiekunów medycznych) oraz pacjentów jadących danym pojazdem w danym czasie:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	vehicles		ALL	PRIMARY				2000	10.00	Using where
1	PRIMARY	transfers		ref	FK_vehicle	FK_vehicle	4	medical_transport.vehicles.Id_vehicle	13	11.11	Using where
1	PRIMARY	users		eq_ref	PRIMARY	PRIMARY	4	medical_transport.transfers.FK_user	1	100.00	
2	UNION	vehicles		ALL	PRIMARY				2000	10.00	Using where
2	UNION	transfers		ref	PRIMARY,FK_vehicle	FK_vehicle	4	medical_transport.vehicles.Id_vehicle	13	11.11	Using where
2	UNION	users_transfers		ref	FK_user,medical_facilities	medical_facilities	4	medical_transport.transfers.Id_trans...	1	100.00	
2	UNION	users		eq_ref	PRIMARY	PRIMARY	4	medical_transport.users_transfers.F...	1	100.00	
3	UNION	vehicles		ALL	PRIMARY				2000	10.00	Using where
3	UNION	transfers		ref	PRIMARY,FK_vehicle	FK_vehicle	4	medical_transport.vehicles.Id_vehicle	13	11.11	Using where
3	UNION	passengers		ref	FK_patient,FK_transfer	FK_transfer	4	medical_transport.transfers.Id_trans...	2	100.00	
3	UNION	patients		eq_ref	PRIMARY	PRIMARY	4	medical_transport.passengers.FK_pa...	1	100.00	
	UNION RESULT	<union1,2,3>		ALL					0		Using temporary

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	vehicles		const	PRIMARY,License_plate_UNIQUE	License_plate_UNIQUE	137	const	1	100.00	Using index
1	PRIMARY	transfers		ref	FK_vehicle_idx	FK_vehicle_idx	4	const	18	11.11	Using where
1	PRIMARY	users		eq_ref	PRIMARY	PRIMARY	4	medical_transport.transfers.FK_user	1	100.00	
2	UNION	vehicles		const	PRIMARY,License_plate_UNIQUE	License_plate_UNIQUE	137	const	1	100.00	Using index
2	UNION	transfers		ref	PRIMARY,FK_vehicle_idx	FK_vehicle_idx	4	const	18	11.11	Using where
2	UNION	users_transfers		ref	FK_transfer_idx,FK_user_idx	FK_transfer_idx	4	medical_transport.transfers.Id_transfer	1	100.00	
2	UNION	users		eq_ref	PRIMARY	PRIMARY	4	medical_transport.users_transfers.FK_user	1	100.00	
3	UNION	vehicles		const	PRIMARY,License_plate_UNIQUE	License_plate_UNIQUE	137	const	1	100.00	Using index
3	UNION	transfers		ref	PRIMARY,FK_vehicle_idx	FK_vehicle_idx	4	const	18	11.11	Using where
3	UNION	passengers		ref	FK_transfer_idx,FK_patient_idx	FK_transfer_idx	4	medical_transport.transfers.Id_transfer	1	100.00	
3	UNION	patients		eq_ref	PRIMARY	PRIMARY	4	medical_transport.passengers.FK_patient	1	100.00	
	UNION RESULT	<union1,2,3>		ALL					0		Using temporary

2. Lista placówek oferujących dane badanie w danym mieście:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	medical_exams		ALL	PRIMARY				350	10.00	Using where
1	SIMPLE	exam_offers		ref	FK_medical_facility,FK_medical_exam	FK_medical_exam	4	medical_transport.medical_exams.Id_medical_exam	6	100.00	
1	SIMPLE	medical_facilities		eq_ref	PRIMARY,FK_location	PRIMARY	4	medical_transport.exam_offers.FK_medical_facility	1	100.00	
1	SIMPLE	locations		eq_ref	PRIMARY	PRIMARY	4	medical_transport.medical_facilities.FK_location	1	10.00	Using where

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	locations		ref	PRIMARY,City_idx	City_idx	137	const	1	100.00	Using index
1	SIMPLE	medical_exams		ref	PRIMARY,Name_idx	Name_idx	137	const	1	100.00	Using index
1	SIMPLE	medical_facilities		ref	PRIMARY,FK_location_idx	FK_location_idx	4	medical_transport.locations.Id_location	2	100.00	
1	SIMPLE	exam_offers		ref	FK_medical_exam_idx,FK_medical_facility_idx	FK_medical_facility_idx	4	medical_transport.medical_facilities.Id_medical_facility	2	1.87	Using where

3. Liczba wolnych i zajętych miejsc na poszczególnych oddziałach w danej placówce:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	hospital_wards		index	PRIMARY,FK_medical_facility	PRIMARY	4		1000	100.00	
1	SIMPLE	medical_facilities		eq_ref	PRIMARY,FK_location	PRIMARY	4	medical_transport.hospital_wards.FK_medical_facility	1	100.00	
1	SIMPLE	locations		eq_ref	PRIMARY	PRIMARY	4	medical_transport.medical_facilities.FK_location	1	10.00	Using where
1	SIMPLE	admissions		ref	FK_hospital_ward	FK_hospital_ward	4	medical_transport.hospital_wards.Id_hospital_ward	33	10.00	Using where

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	locations		ref	PRIMARY,City_idx	City_idx	137	const	1	100.00	Using index; Using temporary
1	SIMPLE	medical_facilities		ref	PRIMARY,FK_location_idx	FK_location_idx	4	medical_transport.locations.Id_location	2	100.00	
1	SIMPLE	hospital_wards		ref	PRIMARY,FK_medical_facility_idx	FK_medical_facility_idx	4	medical_transport.medical_facilities.Id_medical_facility	2	100.00	
1	SIMPLE	admissions		ref	FK_hospital_ward	FK_hospital_ward	4	medical_transport.hospital_wards.Id_hospital_ward	33	10.00	Using where

4. Liczba skierowań wypisanych na dane badanie w konkretnym okresie:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	medical_exams		ALL	PRIMARY				350	10.00	Using where
1	SIMPLE	referrals_medical_exams		ref	FK_medical_exam,FK_referral	FK_medical_exam	4	medical_transport.medical_exams.Id_medical_exam	40	100.00	
1	SIMPLE	referrals		eq_ref	PRIMARY	PRIMARY	4	medical_transport.referrals_medical_exams.FK_referral	1	11.11	Using where

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	medical_exams		ref	PRIMARY,Name_idx	Name_idx	137	const	1	100.00	Using index
1	SIMPLE	referrals_medical_exams		ref	FK_medical_exam_idx,FK_referral_idx	FK_medical_exam_idx	4	medical_transport.medical_exams.Id_medical_exam	40	100.00	
1	SIMPLE	referrals		eq_ref	PRIMARY,Referral_date_idx	PRIMARY	4	medical_transport.referrals_medical_exams.FK_referral	1	11.11	Using where

5. Lista pacjentów najczęściej korzystających z przejazdów (dziesięciu pacjentów):

W kolumnie Extra występuje “Using index” ponieważ zapytanie wykorzystuje klucz główny do odczytania informacji z tabeli – klucz główny funkcjonuje jako indeks, ale nie można go usunąć z bazy.

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	patients		index	PRIMARY	PRIMARY	4		14922	100.00	Using temporary; Using filesort
1	SIMPLE	passengers		ref	FK_patient	FK_patient	4	medical_transport.patients.Id_patie...	1	100.00	Using index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	patients		index	PRIMARY	PRIMARY	4		14940	100.00	Using temporary; Using filesort
1	SIMPLE	passengers		ref	FK_patient_idx	FK_patient_idx	4	medical_transport.patients.Id_patient	1	100.00	Using index

## 6. Lista pracowników zatrudnionych w danej placówce z informacją o stażu pracy każdego pracownika:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	employments		ref	FK_medical_facility,FK_user				350	10.00	Using where; Using filesort
1	SIMPLE	medical_facilities		eq_ref	PRIMARY	PRIMARY	4	medical_transport.employments.FK_medical_facility	1	10.00	Using where
1	SIMPLE	users		eq_ref	PRIMARY	PRIMARY	4	medical_transport.employments.FK_user	1	100.00	

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	medical_facilities		ref	PRIMARY,Name_idx	Name_idx	137	const	1	100.00	Using index; Using temporary; Using filesort
1	SIMPLE	employments		ref	FK_user_idx,FK_medical_facility	FK_medical_facility	4	medical_transport.medical_facilities.Id_medical_facility	1	10.00	Using where
1	SIMPLE	users		eq_ref	PRIMARY	PRIMARY	4	medical_transport.employments.FK_user	1	100.00	

## 7. Średnia liczba przejazdów kierowców:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<derived2>		ALL					31556	100.00	
2	DERIVED	transfers		index	FK_user	FK_user	4		13962	100.00	Using index; Using temporary; Using filesort
2	DERIVED	users		eq_ref	PRIMARY	PRIMARY	4	medical_transport.transfers.FK_user	1	100.00	
2	DERIVED	roles		ref	FK_user	FK_user	4	medical_transport.transfers.FK_user	2	100.00	Using index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	<derived2>		ALL					30716	100.00	
2	DERIVED	transfers		index	FK_user_idx	FK_user_idx	4		13962	100.00	Using index; Using temporary; Using filesort
2	DERIVED	users		eq_ref	PRIMARY>Login_UNIQUE	PRIMARY	4	medical_transport.transfers.FK_user	1	100.00	
2	DERIVED	roles		ref	FK_user_UNIQUE	FK_user_UNIQUE	4	medical_transport.transfers.FK_user	2	100.00	Using index

## 8. Lista placówek z informacją dotyczącą liczby wykonanych z nich przejazdów w danym okresie:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	medical_facilities		index	PRIMARY,FK_location,FK_user	PRIMARY	4		1000	100.00	Using temporary; Using filesort
1	SIMPLE	transfers		ref	FK_facility_to	FK_facility_to	4	medical_transport.medical_facilities.Id_medical_facility	27	100.00	Using index

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	medical_facilities		index	PRIMARY,FK_location_idx,Name_idx,FK_user	PRIMARY	4		1000	100.00	Using temporary; Using filesort
1	SIMPLE	transfers		ref	FK_facility_to_idx	FK_facility_to_idx	4	medical_transport.medical_facilities.Id_medical_facility	27	100.00	Using index

## 9. Placówka z największym dziennym limitem badań na każde badanie:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	exam_offers		ALL					1200	100.00	Using temporary
1	SIMPLE	medical_exams		eq_ref	PRIMARY	PRIMARY	4	medical_transport.exam_offers.FK_medical_exam	1	100.00	
1	SIMPLE	medical_facilities		eq_ref	PRIMARY	PRIMARY	4	medical_transport.exam_offers.FK_medical_facility	1	100.00	

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	exam_offers		ALL					1200	100.00	Using temporary
1	SIMPLE	medical_exams		eq_ref	PRIMARY,Name_idx	PRIMARY	4	medical_transport.exam_offers.FK_medical_exam	1	100.00	
1	SIMPLE	medical_facilities		eq_ref	PRIMARY	PRIMARY	4	medical_transport.exam_offers.FK_medical_facility	1	100.00	

## 10. Lista różnych oddziałów w danym mieście:

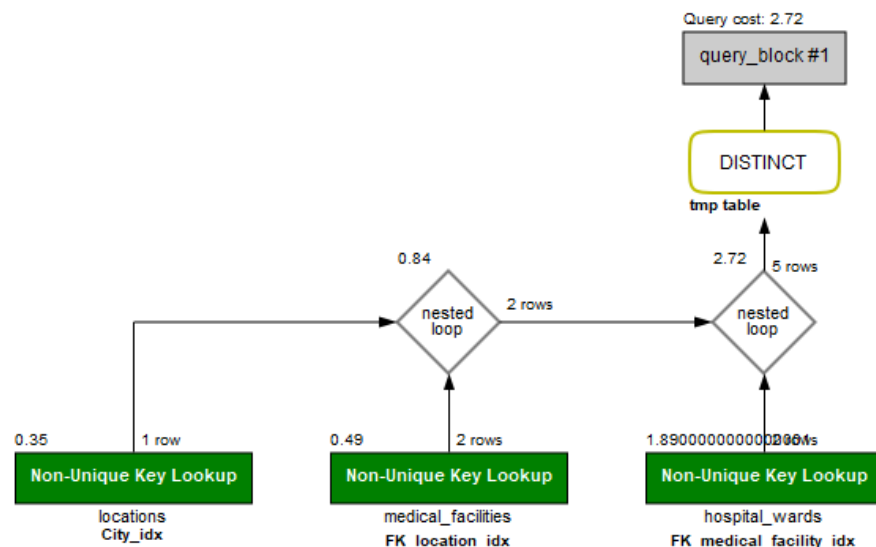
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	medical_facilities		index	PRIMARY,FK_location	FK_location	4		1000	100.00	Using where; Using index; Using temporary
1	SIMPLE	locations		eq_ref	PRIMARY	PRIMARY	4	medical_transport.medical_facilities.FK_location	1	10.00	Using where
1	SIMPLE	hospital_wards		ref	FK_medical_facility	FK_medical_facility	4	medical_transport.medical_facilities.Id_medical_facility	2	100.00	

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	locations		ref	PRIMARY,City_idx	City_idx	137	const	1	100.00	Using index; Using temporary
1	SIMPLE	medical_facilities		ref	PRIMARY,FK_location_idx	FK_location_idx	4	medical_transport.locations.Id_location	2	100.00	Using where; Using index
1	SIMPLE	hospital_wards		ref	FK_medical_facility_idx	FK_medical_facility_idx	4	medical_transport.medical_facilities.Id_medical_facility	2	100.00	

Dzięki użyciu indeksów działanie kwerend jest szybsze – kolumna rows pokazuje że średnio potrzeba sprawdzić mniejszą ilość wierszy żeby odnaleźć potrzebne dane. Nie trzeba też sprawdzać wszystkich wierszy w tabeli (ALL w kwerendzie 4). Indeksy pozwalają w tabeli EXTRA zamienić using where na using indeks – nie trzeba filtrować otrzymanych wyników a jedynie używać indeksów (indeks zamiast odczytu z tabeli).

## Visual Explain:



Czy można inaczej zdefiniować zapytania?



Krzysztof Bosak  
@krzysztofbosak

...

1. Można. Gdyby to było złe to Bóg by inaczej świat stworzył.

:)

Optymalizacja zapytań:

1. W przypadku pisania zapytań pod istniejącą bazę danych należy wykorzystywać indeksy zdefiniowane w bazie. W naszej sytuacji po prostu utworzyliśmy indeksy, które były potem wykorzystywane w kwerendach.
2. Optymalizacja kwerend bez użycia indeksów:  
<https://www.sqlpedia.pl/zlozonosc-obliczeniowa-zapytania-do-duzych-tabel/>  
JOIN nie działa efektywnie dla dużej liczby rekordów, więc należy wykorzystać rozwiązanie nie używające JOIN. W znalezionym przykładzie wykorzystano Common Table Expressions oraz funkcję OVER().
3. Wskazówki do optymalizacji kwerend:  
<https://www.analyticsvidhya.com/blog/2021/10/a-detailed-guide-on-sql-query-optimization/>
  - a. unikać wykorzystania correlated nested queries – SELECT wewnątrz SELECT,
  - b. unikać wykorzystania INNER JOIN z warunkami równości lub OR,
  - c. używać wildcards – \* – tylko tam gdzie są niezbędne,
  - d. sprawdzać czy rekordy istnieją, zanim pobierzemy je z bazy danych.

- e. używać WHERE zamiast HAVING tam gdzie to możliwe – wykorzystywać HAVING tylko dla wyników funkcji agregujących i w grupowaniu,
- f. wykorzystywać odpowiednio indeksy i tworzyć indeksy wielokolumnowe.

## Indeksy

Indeksy umożliwiają kwerendom efektywne wyszukiwanie i pobieranie danych z bazy danych. Indeksy są tworzone dla konkretnych tabel, z tym że tabela może mieć więcej niż jeden indeks. Indeksy składają się z kluczy, które opierają się na kolumnach tabeli – klucze odpowiadają wartościom, które chcemy wyszukiwać w danym indeksie. Poprzez porównywanie kluczy do indeksu możliwe jest znalezienie odpowiednich rekordów w bazie danych.

Indeksy działają trochę jak spis treści w książce posortowany według jednej wartości, z numerami stron prowadzącymi do pozostałych informacji.

Zasadniczą zaletą indeksów jest przyspieszenie działania kwerend – taka też jest ich główna funkcja. Niestety, przechowywanie indeksów zużywa zasoby pamięciowe, a także przy dodawaniu czy edycji danych indeksy muszą być odpowiednio aktualizowane, co zajmuje trochę czasu.

Typy struktur indeksów:

- B-Tree:
  - łatwe do utrzymania, skalowania czy powiększania,
  - mogą być używane do porównań używających =, >, >=, <, <=, lub BETWEEN,
  - używany w większości przypadków, często jako default,
  - oferowane przez praktycznie wszystkie serwery bazodanowe.
- Hash:
  - wyszukiwanie szybsze od B-tree,
  - dostęp do elementów jedynie poprzez ich PRIMARY KEY,
  - używane tylko do porównań równościowych, używających = lub ⇔,
  - nie można wykonywać porównań zwracających zakresy np. <, >,
  - nie mogą zostać wykorzystane do sortowania danych,
  - jedynie całe klucze mogą być wykorzystywane do wyszukiwania wiersza.
- R-Tree:
  - używane głównie przy danych przestrzennych i N-wymiarowych.

Wady i zalety typów struktur indeksów:

- B-Tree:
  - bardzo uniwersalne zastosowanie z uwagi na wiele dostępnych porównań, możliwość porównań zwracających zakresy, możliwość sortowania danych oraz możliwość wykorzystywania częściowych kluczy do wyszukiwania wiersza,

- trochę wolniejsze wyszukiwanie, konieczność odpowiedniego utrzymywania drzewa spowalnia dodawanie i edycję indeksów.
- Hash:
  - bardzo ograniczone zastosowanie, przez co Hash jest strukturą indeksu rzadko używaną przez serwery bazodanowe (jest używana tylko w wyjątkowych przypadkach),
  - wyszukiwanie szybsze od B-tree oraz brak konieczności odpowiedniego utrzymywania drzewa.
- R-Tree:
  - bardzo uniwersalne zastosowanie z uwagi na wiele dostępnych porównań, możliwość porównań zwracających zakresy, możliwość sortowania danych oraz możliwość wykorzystywania częściowych kluczy do wyszukiwania wiersza, dodatkowo R-tree jest zoptymalizowane pod używanie danych przestrzennych i N-wymiarowych,
  - trochę wolniejsze wyszukiwanie, konieczność odpowiedniego utrzymywania drzewa spowalnia dodawanie i edycję indeksów.

Źródło:

[https://www.oreilly.com/library/view/high-performance-mysql/0596003064/ch04.html?fbclid=IwAR0Ag37vFWv\\_QuO\\_X51Zjd4zkG6vRDPdGsyku\\_NG0woMdvo\\_VpZM\\_2812FU](https://www.oreilly.com/library/view/high-performance-mysql/0596003064/ch04.html?fbclid=IwAR0Ag37vFWv_QuO_X51Zjd4zkG6vRDPdGsyku_NG0woMdvo_VpZM_2812FU)

Typy indeksów w MySQL:

- PRIMARY KEY:
  - wartości indeksu muszą być unikatowe,
  - wartości indeksu nie mogą przyjmować wartości NULL,
  - po ustawieniu wartości indeksu nie mogą zostać zmienione.
- SIMPLE INDEX:
  - wartości indeksu nie muszą być unikatowe,
  - wartości indeksu mogą przyjmować wartość NULL.
- UNIQUE INDEX:
  - wartości indeksu muszą być unikatowe,
  - wartości indeksu mogą przyjmować wartości NULL,
  - dla indeksu jednokolumnowego wartości w kolumnie nie mogą się powtarzać,
  - dla indeksu wielokolumnowego wartości w poszczególnych kolumnach mogą się powtarzać, ale w każdym wierszu kombinacja wartości kolumn indeksu musi być unikatowa.
- FULLTEXT INDEX:
  - wykorzystywany jest do wyszukiwania pełnotekstowego (full-text search) – wykorzystywany jest najczęściej w bazach danych przechowujących pełne dokumenty tekstowe.
  - zamiast indeksować całą wartość (duży dokument tekstowy) indeksowane są słowa w bloku tekstowym – umożliwia to przyspieszenie wyszukiwania konkretnych słów i wyrażeń w teście.

- DESCENDING INDEX:
  - o zwykły indeks, ale przechowywany w kolejności malejącej,
  - o używany jest gdy kwerenda zażąda kolejności malejącej, np. podczas wyszukiwania ostatnio dodanych danych,
  - o MySQL potrafi przechodzić indeksy “w odwrotnej kolejności” jeśli jest to konieczne – z tego względu niepotrzebne jest używanie DESCENDING INDEX. Przejście indeksu “w odwrotnej kolejności” jest jednak mniej efektywne niż użycie DESCENDING INDEX.
- SPATIAL INDEX:
  - o wykorzystywany do indeksowania kolumn przechowujących wartości przestrzenne, np. wartości geometrycznych lub geograficznych,
  - o ze względu na wąski zakres wykorzystania nie przyda się w naszym projekcie.

Źródła:

<https://vanseodesign.com/web-design/the-types-of-indexes-you-can-add-to-mysql-tables/>

<https://dev.mysql.com/doc/refman/8.0/en/descending-indexes.html>

[https://www.oreilly.com/library/view/high-performance-mysql/0596003064/ch04.html?fbclid=IwAR0Ag37vFWv\\_OuO\\_X51Zid4zkG6vRDPdGsyku\\_NG0woMdvo\\_VpZM\\_2812FU](https://www.oreilly.com/library/view/high-performance-mysql/0596003064/ch04.html?fbclid=IwAR0Ag37vFWv_OuO_X51Zid4zkG6vRDPdGsyku_NG0woMdvo_VpZM_2812FU)

MySQL zawsze wykorzysta tylko jeden indeks z danej kolumny przy optymalizacji wykonania kwerendy.

## Skrypt SQL po dodaniu indeksów

```
-- MySQL Script generated by MySQL Workbench
-- Tue Oct 26 17:58:26 2021
-- Model: New Model      Version: 1.0
-- MySQL Workbench Forward Engineering

--
-----
-- Schema medical_transport
--
-----
drop schema medical_transport;

--
-----
-- Schema medical_transport
--
-----
CREATE SCHEMA IF NOT EXISTS `medical_transport` DEFAULT CHARACTER SET utf8 ;
USE `medical_transport` ;

--
-----
-- Table `medical_transport`.`Users`
--
-----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Users` (
  `Id_user` INT NOT NULL AUTO_INCREMENT,
  `Login` VARCHAR(45) NOT NULL,
  `Password` VARCHAR(45) NOT NULL,
  `Name` VARCHAR(45) NOT NULL,
  `Surname` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id_user`),
  UNIQUE INDEX `Login_UNIQUE` (`Login` ASC) VISIBLE)
ENGINE = InnoDB;
```

```

-----
-- Table `medical_transport`.`Locations`
-----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Locations` (
  `Id_location` INT NOT NULL AUTO_INCREMENT,
  `City` VARCHAR(45) NOT NULL,
  `Address` VARCHAR(45) NOT NULL,
  `Postal_code` VARCHAR(6) NOT NULL,
  PRIMARY KEY (`Id_location`),
  INDEX `City_idx` (`City` ASC) VISIBLE)
ENGINE = InnoDB;

-----
-- Table `medical_transport`.`Medical_facilities`
-----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Medical_facilities` (
  `Id_medical_facility` INT NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(45) NOT NULL,
  `FK_location` INT NOT NULL,
  `FK_user` INT NOT NULL,
  PRIMARY KEY (`Id_medical_facility`),
  INDEX `FK_location_idx` (`FK_location` ASC) VISIBLE,
  INDEX `Name_idx` (`Name` ASC) VISIBLE,
  CONSTRAINT
    FOREIGN KEY (`FK_location`)
      REFERENCES `medical_transport`.`Locations` (`Id_location`),
  CONSTRAINT
    FOREIGN KEY (`FK_user`)
      REFERENCES `medical_transport`.`Users` (`Id_user`))
ENGINE = InnoDB;

-----
-- Table `medical_transport`.`Hospital_wards`
-----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Hospital_wards` (
  `Id_hospital_ward` INT NOT NULL AUTO_INCREMENT,
  `Name` VARCHAR(65) NOT NULL,
  `Places` INT NOT NULL,
  `FK_medical_facility` INT NOT NULL,
  PRIMARY KEY (`Id_hospital_ward`),
  INDEX `FK_medical_facility_idx` (`FK_medical_facility` ASC) VISIBLE,
  CONSTRAINT
    FOREIGN KEY (`FK_medical_facility`)
      REFERENCES `medical_transport`.`Medical_facilities`
        (`Id_medical_facility`))
ENGINE = InnoDB;

-----
-- Table `medical_transport`.`Vehicles`
-----

```



```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Vehicles` (
  `Id_vehicle` INT NOT NULL AUTO_INCREMENT,
  `Brand` VARCHAR(45) NOT NULL,
  `Model` VARCHAR(45) NOT NULL,
  `License_plate` VARCHAR(45) NOT NULL,
  `Seats` INT NOT NULL,
  `FK_medical_facility` INT NOT NULL,
  PRIMARY KEY (`Id_vehicle`),
  CONSTRAINT CHECK(`Seats`>=0),
  UNIQUE INDEX `License_plate_UNIQUE` (`License_plate` ASC) VISIBLE,
  CONSTRAINT `FK_medical_facility`
  FOREIGN KEY (`FK_medical_facility`)
  REFERENCES `medical_transport`.`Medical_facilities`
  (`Id_medical_facility`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Transfers`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Transfers` (
  `Id_transfer` INT NOT NULL AUTO_INCREMENT,
  `Start_time` TIME NOT NULL,
  `Start_date` DATE NOT NULL,
  `FK_user` INT NOT NULL,
  `FK_vehicle` INT NOT NULL,
  `FK_facility_from` INT NOT NULL,
  `FK_facility_to` INT NOT NULL,
  PRIMARY KEY (`Id_transfer`),
  INDEX `FK_user_idx` (`FK_user` ASC) VISIBLE,
  INDEX `FK_vehicle_idx` (`FK_vehicle` ASC) VISIBLE,
  INDEX `FK_facility_to_idx` (`FK_facility_to` ASC) VISIBLE,
  CONSTRAINT
    FOREIGN KEY (`FK_user`)
    REFERENCES `medical_transport`.`Users` (`Id_user`),
  CONSTRAINT
    FOREIGN KEY (`FK_vehicle`)
    REFERENCES `medical_transport`.`Vehicles` (`Id_vehicle`),
  CONSTRAINT `FK_facility_from`
    FOREIGN KEY (`FK_facility_from`)
    REFERENCES `medical_transport`.`Medical_facilities`
    (`Id_medical_facility`),
  CONSTRAINT `FK_facility_to`
    FOREIGN KEY (`FK_facility_to`)
    REFERENCES `medical_transport`.`Medical_facilities`
    (`Id_medical_facility`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Patients`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Patients` (

```

```

`Id_patient` INT NOT NULL AUTO_INCREMENT,
`Name` VARCHAR(45) NOT NULL,
`Surname` VARCHAR(45) NOT NULL,
PRIMARY KEY (`Id_patient`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Admissions`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Admissions` (
  `Id_admission` INT NOT NULL AUTO_INCREMENT,
  `FK_hospital_ward` INT NOT NULL,
  `FK_patient` INT NOT NULL,
  `Admission_date` DATE NOT NULL,
  `Discharge_date` DATE,
  PRIMARY KEY (`Id_admission`),
  INDEX `FK_patient_idx` (`FK_patient` ASC) VISIBLE,
  INDEX `Admission_date_idx` (`Admission_date` ASC) VISIBLE,
  CONSTRAINT CHECK(`Admission_date`<=`Discharge_date`),
  CONSTRAINT
    FOREIGN KEY (`FK_hospital_ward`)
    REFERENCES `medical_transport`.`Hospital_wards` (`Id_hospital_ward`),
  CONSTRAINT
    FOREIGN KEY (`FK_patient`)
    REFERENCES `medical_transport`.`Patients` (`Id_patient`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Referrals`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Referrals` (
  `Id_referral` INT NOT NULL AUTO_INCREMENT,
  `FK_user` INT NOT NULL,
  `FK_patient` INT NOT NULL,
  `Referral_date` DATE NOT NULL,
  `Body_part` VARCHAR(45) NULL,
  PRIMARY KEY (`Id_referral`),
  INDEX `Referral_date_idx` (`Referral_date` ASC) VISIBLE,
  CONSTRAINT
    FOREIGN KEY (`FK_user`)
    REFERENCES `medical_transport`.`Users` (`Id_user`),
  CONSTRAINT
    FOREIGN KEY (`FK_patient`)
    REFERENCES `medical_transport`.`Patients` (`Id_patient`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Medical_exams`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Medical_exams` (

```

```

`Id_medical_exam` INT NOT NULL AUTO_INCREMENT,
`Name` VARCHAR(45) NOT NULL,
PRIMARY KEY (`Id_medical_exam`),
INDEX `Name_idx` (`Name` ASC) VISIBLE)
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Reservations`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Reservations` (
  `Id_reservation` INT NOT NULL AUTO_INCREMENT,
  `Start_date` DATE NOT NULL,
  `Start_time` TIME NOT NULL,
  `FK_patient` INT NOT NULL,
  `FK_medical_exam` INT NOT NULL,
  PRIMARY KEY (`Id_reservation`),
  CONSTRAINT
    FOREIGN KEY (`FK_patient`)
    REFERENCES `medical_transport`.`Patients` (`Id_patient`),
  CONSTRAINT
    FOREIGN KEY (`FK_medical_exam`)
    REFERENCES `medical_transport`.`Medical_exams` (`Id_medical_exam`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Roles`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Roles` (
  `Id_role` INT NOT NULL AUTO_INCREMENT,
  `FK_user` INT NOT NULL,
  `Name` ENUM("Adm", "Sup", "AEm", "Dri", "Doc", "Med") NOT NULL,
  PRIMARY KEY (`Id_role`),
  INDEX `FK_user_UNIQUE` (`FK_user` ASC) VISIBLE,
  CONSTRAINT
    FOREIGN KEY (`FK_user`)
    REFERENCES `medical_transport`.`Users` (`Id_user`))
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Exam_offers`
-- -----

```

```

CREATE TABLE IF NOT EXISTS `medical_transport`.`Exam_offers` (
  `Id_exam_offer` INT NOT NULL AUTO_INCREMENT,
  `FK_medical_facility` INT NOT NULL,
  `FK_medical_exam` INT NOT NULL,
  `Daily_limit` INT NULL,
  PRIMARY KEY (`Id_exam_offer`),
  CONSTRAINT CHECK(`Daily_limit`>=0),
  INDEX `FK_medical_exam_idx` (`FK_medical_exam` ASC) VISIBLE,
  INDEX `Daily_limit_idx` (`Daily_limit` ASC) VISIBLE,

```

```

INDEX `FK_medical_facility_idx` (`FK_medical_facility` ASC) VISIBLE,
CONSTRAINT
    FOREIGN KEY (`FK_medical_facility`)
    REFERENCES `medical_transport`.`Medical_facilities`
    (`Id_medical_facility`),
CONSTRAINT
    FOREIGN KEY (`FK_medical_exam`)
    REFERENCES `medical_transport`.`Medical_exams` (`Id_medical_exam`)
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Employments`
-- -----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Employments` (
    `Id_employment` INT NOT NULL AUTO_INCREMENT,
    `FK_medical_facility` INT NOT NULL,
    `FK_user` INT NOT NULL,
    `Employment_date` DATE NOT NULL,
    `Dismissal_date` DATE NULL,
    PRIMARY KEY (`Id_employment`),
    CONSTRAINT CHECK(`Employment_date` <= `Dismissal_date`),
    INDEX `FK_user_idx` (`FK_user` ASC) VISIBLE,
    INDEX `Employment_date_idx` (`Employment_date` ASC) VISIBLE,
    CONSTRAINT
        FOREIGN KEY (`FK_medical_facility`)
        REFERENCES `medical_transport`.`Medical_facilities`
        (`Id_medical_facility`),
    CONSTRAINT
        FOREIGN KEY (`FK_user`)
        REFERENCES `medical_transport`.`Users` (`Id_user`)
ENGINE = InnoDB;

```

```

-- -----
-- Table `medical_transport`.`Passengers`
-- -----
CREATE TABLE IF NOT EXISTS `medical_transport`.`Passengers` (
    `Id_passenger` INT NOT NULL AUTO_INCREMENT,
    `FK_patient` INT NOT NULL,
    `FK_transfer` INT NOT NULL,
    `Needs_care` ENUM('Yes', 'No') NOT NULL,
    `Status` ENUM('Lying', 'Sitting', 'Walking') NOT NULL,
    INDEX `FK_transfer_idx` (`FK_transfer` ASC) VISIBLE,
    INDEX `FK_patient_idx` (`FK_patient` ASC) VISIBLE,
    PRIMARY KEY (`Id_passenger`),
    CONSTRAINT
        FOREIGN KEY (`FK_patient`)
        REFERENCES `medical_transport`.`Patients` (`Id_patient`),
    CONSTRAINT
        FOREIGN KEY (`FK_transfer`)
        REFERENCES `medical_transport`.`Transfers` (`Id_transfer`)
        ON DELETE CASCADE)

```

```
ENGINE = InnoDB;
```

```
-- -----  
-- Table `medical_transport`.`Refferals_Medical_exams`  
-- -----
```

```
CREATE TABLE IF NOT EXISTS `medical_transport`.`Refferals_Medical_exams` (  
  `Id_ref_med` INT NOT NULL AUTO_INCREMENT,  
  `FK_medical_exam` INT NOT NULL,  
  `FK_referral` INT NOT NULL,  
  PRIMARY KEY (`Id_ref_med`),  
  INDEX `FK_medical_exam_idx` (`FK_medical_exam` ASC) VISIBLE,  
  INDEX `FK_referral_idx` (`FK_referral` ASC) VISIBLE,  
  CONSTRAINT  
    FOREIGN KEY (`FK_medical_exam`)  
    REFERENCES `medical_transport`.`Medical_exams` (`Id_medical_exam`),  
  CONSTRAINT  
    FOREIGN KEY (`FK_referral`)  
    REFERENCES `medical_transport`.`Referrals` (`Id_referral`))  
ENGINE = InnoDB;
```

```
-- -----  
-- Table `medical_transport`.`Users_Transfers`  
-- -----
```

```
CREATE TABLE IF NOT EXISTS `medical_transport`.`Users_Transfers` (  
  `Id_user_transfer` INT NOT NULL AUTO_INCREMENT,  
  `FK_user` INT NOT NULL,  
  `FK_transfer` INT NOT NULL,  
  PRIMARY KEY (`Id_user_transfer`),  
  INDEX `FK_transfer_idx` (`FK_transfer` ASC) VISIBLE,  
  INDEX `FK_user_idx` (`FK_user` ASC) VISIBLE,  
  CONSTRAINT  
    FOREIGN KEY (`FK_user`)  
    REFERENCES `medical_transport`.`Users` (`Id_user`),  
  CONSTRAINT medical_facilities  
    FOREIGN KEY (`FK_transfer`)  
    REFERENCES `medical_transport`.`Transfers` (`Id_transfer`)  
    ON DELETE CASCADE)  
ENGINE = InnoDB;
```

## **Etap 7.**

### **Wnioski dotyczące realizacji projektu**

- Przede wszystkim, przyjęto założenie “Baza nie służy do przechowywania dokumentacji medycznej czy osobistej pacjentów i pracowników – system ma działać na minimalnej ilości danych wystarczających do identyfikacji danej osoby.” To założenie znacząco ogranicza zakres projektu do zakresu wykonalnego w cztery osoby w pół semestru, ale ogranicza także przydatność projektowanego systemu. W rzeczywistości taki system byłby przydatny jedynie zintegrowany z systemem administracyjnym placówek medycznych.
- Początkowo projekt bazy nie przewidywał weryfikowania czy użytkownicy byli przydzielani do odpowiednich dla nich zadań (bazując na informacjach o ich rolach). W związku z tym pojawiały się nieścisłości i absurdy w przechowywaniu danych (lekarz przypisany do transportu jako kierowca). Dlatego konieczne było wprowadzenie dodatkowych ograniczeń przy generowaniu danych. Należałoby również wprowadzić dodatkowe ograniczenia w strukturze bazy danych, aby nie mogły wystąpić takie sytuacje z powodu błędu wprowadzania użytkownika.
- Przygotowując projekt bazy danych należy poświęcić dużo czasu na analizę wycinka rzeczywistości i odpowiednie zrozumienie jakie informacje i dlaczego są potrzebne. W idealnej sytuacji wszystkie atrybuty wszystkich encji zostałyby poprawnie zdefiniowane w etapie 2. W naszym projekcie w późniejszych etapach należało dodać do transportu informację z i do jakiej placówki prowadzi. Ta informacja powinna była pojawić się już w etapie 2. Dodano też informację o głównej placówce pojazdu, aby baza zawierała więcej przydatnych informacji.
- Aby uprościć przechowywanie danych użytkowników w bazie ważne jest posiadanie jednej tabeli z użytkownikami i drugiej z ich rolami – był to jeden z błędów poprawionych po oddaniu 1 etapu. Gdyby istniało wiele tabel użytkowników (osobna tabela dla każdej roli) po pierwsze jedna osoba musiałaby mieć wiele kont, jeśli powinna posiadać uprawnienia wielu użytkowników, oraz po drugie przy logowaniu należałoby przeszukać wszystkie tabele użytkowników celem identyfikacji odpowiedniego użytkownika.
- Istotne jest ustalenie i trzymanie się sensownej konwencji nazewnictwa. Wykorzystałyśmy konwencję nazewnictwa zaproponowaną przez prowadzącego i gdyby nie to prawdopodobnie pojawiłyby się problemy z identyfikacją tabel i atrybutów w późniejszych etapach. W etapie 2 klucze obce oznaczałyśmy znakiem # – zamiast tego zaproponowano dodanie przedrostka FK do nazwy atrybutów będących kluczami obcymi, co ułatwiło pracę ze skryptem w Pythonie. Nazwanie tabel zawierających tylko klucze obce i ewentualnie klucz sztuczny przez połączenie nazw tabel do których te atrybuty prowadzą pozwoliło również uniknąć wymyślania synonimicznych nazw, nie oddających celu tabel.
- Wprowadzenie indeksów przyspiesza wyszukiwanie danych w bazie, wymaga jednak przemyślenia, gdzie te indeksy będą najbardziej przydatne. Nie należy tworzyć

indeksów, które nigdy nie będą używane, ponieważ zajmują one miejsce oraz trzeba je utrzymywać przy aktualizacji, dodawaniu i usuwaniu danych. W początkowej realizacji projektu wykorzystano tylko indeksy wygenerowane przy generacji skryptu SQL – po analizie indeksów zostały one w większości usunięte, ponieważ nie były potrzebne.

- Tworzenie interfejsów pozwala lepiej uświadomić sobie potrzeby bazy. Pomagają one nam zrozumieć i wczuć się w docelowego użytkownika korzystającego z bazy danych, dzięki czemu możemy ulepszyć jej wydajność i funkcjonalność. Dokładniejsze zrozumienie potrzeb i problemów, z którymi mierzyć się będzie musiał użytkownik daje nam możliwość lepszego dobrania i usprawnienia tworzonych kwerend, np. poprzez zastosowanie indeksów.
- Początkowo w bazie wykorzystano wiele kluczy złożonych, których należy unikać. Klucze złożone należy zastąpić sztucznymi kluczami w celu przyspieszenia wyszukiwania danych.
- Baza była tworzona w przemyślany sposób – starano się od razu zapewnić normalizację. Dzięki świadomemu tworzeniu tabel w bazie, nie wystąpiła konieczność normalizacji schematu do trzeciej postaci normalnej, ponieważ był on w niej od razu.
- Użycie funkcji Check zabezpiecza przed dodaniem do bazy niepoprawnych danych, co zwiększa stopień bezpieczeństwa. W bazie użyto jej jednak tylko w kilku przypadkach. Funkcja CHECK posiada mocno ograniczone zastosowania. Nie można korzystać z funkcji niedeterministycznych, czyli np. porównywać dat dodawanych do bazy z datą obecną, co chcieliśmy zrobić w celu ograniczenia możliwości wprowadzania błędnych dat przez użytkowników.

## **Analiza SWOT:**

### **Mocne strony:**

- Poprawnie wykonany produkt, gwarantujący funkcjonowanie bez anomalii.
- Przechowywanie informacji o placówkach w całym kraju, a nie tylko danym regionie.
- Instynktowny interfejs systemu dostępowego do zaprojektowanej bazy.

### **Słabe strony:**

- Brak połączenia bazy z danymi szpitali – przydatność systemu jest ograniczona ze względu na ograniczenia nałożone na zakres projektu. Nasz system bazodanowy byłby faktycznie przydatny zintegrowany z systemami placówek.
- Konieczność dostosowania bazy do różnych systemów działających w placówkach, w przypadku decyzji o integracji bazy w działające już systemy.

### **Szanse:**

- Rosnący popyt na usługi transportowe pacjentów pomiędzy szpitalami, wynikający z obecnej sytuacji pandemicznej.
- Wspomaganie i usprawnianie organizacji transportu pacjentów pomiędzy szpitalami.
- Brak konkurencji na rynku – obecnie nie istnieje system, który pełniłby rolę podobną do zaprojektowanego systemu.

**Zagrożenia:**

- Brak zaufania osób z branży medycznej do systemów informatycznych – wiele lekarzy, lekarek, pielęgniarzy i pielęgniarek uważa, że systemy informatyczne są przygotowywane przez osoby, które nie rozumieją realnych potrzeb i problemów w branży medycznej. Niechętnie korzystają z wdrożonych już systemów i są zmuszeni z korzystania z wielu niezintegrowanych ze sobą systemów.

*źródło: mama Moniki G. (pielęgniarka), mama Aleksandry S. (lekarz rodzinny)*

- Możliwe rozbieżności danych co do faktycznego stanu np. wolnych miejsc na oddziałach.
- Trudności personelu w posługiwaniu się systemem i związane z tym nieścisłości lub błędy w przechowywanych danych.

**Kierunki rozwoju**

- Połączenie bazy z bazą pacjentów szpitali – eliminuje jedną ze słabych stron i jedno z zagrożeń.
- Stworzenie do bazy aplikacji webowej i mobilnej.
- Wprowadzenie możliwości obsługi przejazdów międzynarodowych – sprowadzenie z zagranicy pacjenta z nagłą chorobą – “transport daleki”.
- Wprowadzenie opcji obliczania kosztów przejazdu.
- Rozszerzenie bazy na placówki prywatne.
- Dostosowanie bazy do sytuacji z COVID-19 np. przechowywanie informacji o szczepieniu, wyniku testu, kwarantannie.



## Etap 8.

Aby rejestrować zmiany danych dotyczących transportów używamy wolnozmiennych wymiarów.

### **Wolnozmiennie wymiary (slowly changing dimensions)**

Teoria hurtowni danych proponuje między innymi dwa podejścia do projektowania hurtowni danych – podejście “wymiarowe” (dimensional) i “normalizacyjne” (normalized). W podejściu wymiarowym dane przechowywane są w tabelach faktów i tabelach wymiarów. W tabelach faktów przechowywane są fakty – czyli konkretne, często liczbowe wartości – bez kontekstu. Dopiero w tabeli wymiarów przechowywane są informacje, które nadają kontekst faktom.

Te dwa podejścia nie są wykluczające się, możliwa jest więc normalizacja hurtowni danych zaprojektowanej podejściem “wymiarowym”. Mimo że w ramach projektu opracowywana jest baza danych, a nie hurtownia danych, można zastosować podejście wolnozmiennych wymiarów.

Podstawowe rodzaje wolnozmiennych wymiarów:

- **typ 0** – brak zmian. Nadane początkowe wartości dla rekordów pozostają bez zmian (brak możliwości edycji). Typ ten najczęściej jest stosowany dla atrybutów które są stałe, niezmiennie (np. data urodzenia).
- **typ 1** – nadpisywanie. W tym przypadku, wartości istniejących już rekordów są nadpisywane nowymi wartościami. Ma to swoją wadę, ponieważ nie przechowujemy historii edytowania poszczególnych atrybutów, ale jest bardzo proste w implementacji. Przed modyfikacją takim właśnie wymiarem była tabela Transfers.
- **typ 2** – “wersjonowanie wierszy” (Row Versioning). W tym typie, gdy zmieniają się wartości dla bieżącego rekordu, jest on oznaczany jako zamknięty (najczęściej poprzez nową kolumnę w tabeli) i wstawiany jest nowy rekord z aktualnymi wartościami. Można również przechowywać informację, od kiedy aktualny jest nowy rekord. W ten sposób możemy przechowywać wszystkie historie zmian danego rekordu. Ten typ nie jest skalowalny – dla bardzo dużej liczby wierszy, których atrybuty bardzo szybko się zmieniają należy dodać ogromną liczbę wierszy.
- **typ 3** – dodanie kolumny zawierającej poprzednią wartość atrybutu – na przykład dla atrybutu Nazwisko dodanie kolumny Poprzednie\_nazwisko. Można również przechowywać informację, do kiedy aktualna jest poprzednia wartość. W ten sposób przechowywane są jedynie aktualna i poprzednia wartość atrybutu, zatem jeżeli wartość zostanie zmieniona dwa razy, utraczona zostanie pierwotna wartość atrybutu. Trudno to więc traktować jako przechowywanie pełnej historii zmian. Nie jest też skalowalna – jeżeli w tabeli jest wiele atrybutów, których wartości mogą się zmieniać, należy dodać wiele dodatkowych kolumn.
- **typ 4** – dodanie tabeli zawierającej część lub wszystkie atrybuty wraz z historią zmian. Ta tabela zawiera część atrybutów, jeśli niektóre z atrybutów głównej tabeli są

stałe. W dodatkowo utworzonej tabeli przechowywana jest historia zmian atrybutów z wykorzystaniem row versioning – jest to tabela “historyczna” (history table). Istnieją dwa podejścia – przechowywanie zmieniających się wartości tylko w tabeli historycznej lub w obu tabelach. Jeżeli dane przechowywane są w obu tabelach przy zmianie wartości atrybutów poprzedni stan atrybutów w tabeli głównej zostaje nadpisany, a poprzedni stan zostaje zapisany do tabeli historycznej. Przy pracy na hurtowni danych, w której wydzielone są tabele faktów i tabele wymiarów, z tabeli głównej do tabeli historycznej można “dostać się” jedynie za pośrednictwem tabeli faktów – tabela faktów zawiera klucze obce do wymiaru głównego i historycznego.

- **typ 5** – jest połączeniem typów 4 i 1 – od typu 4 różni się tym, że poza tabelą historyczną tworzona jest dodatkowo tabela przechowująca aktualny stan tabeli. Ma sens dla pierwszego podejścia do etapu 4 – przechowywanie zmieniających się atrybutów tylko w tabeli historycznej. Wtedy aby wyeliminować konieczność odwołania się do aktualnego stanu zmieniających się danych za pośrednictwem faktów, główna tabela zawiera klucz obcy do tabeli przechowującej aktualny stan zmieniających się wartości. Ze względu na fakt, że nie podmiotem projektu nie jest hurtownia danych i nie wprowadzono podziału na tabele faktów i wymiarów, typy 4 i 5 nie są rozróżnialne w implementacji dla projektowanej bazy danych.
- **typ 6** – jest połączeniem typów 1, 2 i 3 – przechowuje on zatem historię zmian poprzez row versioning, ale dodatkowo zawiera kolumnę z poprzednią wartością danej kolumny. Podczas edycji danej kolumny, obecna wartość jest aktualizowana w każdym rekordzie przechowującym historyczne dane dla wybranego obiektu.

Źródła:

<https://www.sqlshack.com/implementing-slowly-changing-dimensions-scds-in-data-warehouses/>

<https://www.datawarehouse4u.info/SCD-Slowly-Changing-Dimensions.html>

<https://www.kimballgroup.com/2013/02/design-tip-152-slowly-changing-dimension-types-0-4-5-6-7/>

[https://en.wikipedia.org/wiki/Slowly\\_changing\\_dimension](https://en.wikipedia.org/wiki/Slowly_changing_dimension)

W ramach modyfikacji mamy przechowywać historię zmian wprowadzanych w transportach np. zmiana czasu wyjazdu, liczby pasażerów, opiekuna medycznego. Aby mieć pełen obraz zmian w transportach należy pamiętać historię zmian dla trzech tabel: Passengers, Transfers oraz Users\_Transfers. Do tego celu użyte zostaną dwa typy SCD – 2 i 5.

Ze względu na konieczność łatwego dostępu dla aktualnych danych dotyczących przewozów i korzystających z nich pacjentów dla tabeli Transfers oraz Passengers wybrano użycie SCD typu 5 – z przystosowaniem go do wykorzystania w bazie danych (bez podziału na tabele faktów i wymiarów). Zatem utworzone zostają tabele History\_Passengers oraz History\_Transfers przechowujące informację o historycznych zmianach, podczas gdy tabele Transfers oraz Passengers zawsze zawierają aktualne dane. Pozwoli to w łatwy sposób uzyskiwać aktualne informacje, ponieważ sięganie do danych historycznych będzie rzadziej wykonywaną operacją.

## Diagram ERD

The diagram is an Entity-Relationship (ER) model for a hospital database. It features several entities represented by rectangles and relationships represented by diamonds. The cardinalities for each relationship are indicated by numbers in parentheses.

- Entities:**
  - ADMINISTRATOR
  - SUPERVISOR
  - MANAGES
  - EMPLOYS
  - DOCTOR
  - ADMINISTRATIVE EMPLOYEE
  - WORKS
  - MEDICAL FACILITY
  - OWNS
  - HOSPITAL WARD
  - POSSESSES
  - PLACE
  - LOCATION
  - IS IN
  - HIRES
  - EXECUTES
  - DRIVER
  - VEHICLE
  - BOOKS
  - TRANSFER
  - OVERSEES
  - MEDIC
  - RESERVATION
  - PARTICIPATES
  - HAS
  - PATIENT
  - GETS
  - REFERRAL
  - ISSUES
  - MEDICAL EXAMINATION
  - IS FOR
- Relationships and Cardinalities:**
  - ADMINISTRATOR (1,1) **CREATES** (0,N) SUPERVISOR
  - SUPERVISOR (1,1) **MANAGES** (0,N) MEDICAL FACILITY
  - DOCTOR (0,N) **EMPLOYS** (0,N) MEDICAL FACILITY
  - ADMINISTRATIVE EMPLOYEE (0,N) **WORKS** (0,1) MEDICAL FACILITY
  - MEDICAL FACILITY (1,1) **OWNS** (0,N) HOSPITAL WARD
  - HOSPITAL WARD (1,1) **POSSESSES** (0,N) PLACE
  - PLACE (0,N) **TAKES** (0,N) PATIENT
  - LOCATION (1,1) **IS IN** (0,1) MEDICAL FACILITY
  - MEDICAL FACILITY (0,1) **HIRES** (0,N) DRIVER
  - DRIVER (1,1) **EXECUTES** (0,N) TRANSFER
  - TRANSFER (0,N) **BOOKS** (1,1) VEHICLE
  - VEHICLE (1,1) **POSSESSES** (0,N) PLACE
  - MEDIC (0,N) **OVERSEES** (0,N) TRANSFER
  - TRANSFER (0,N) **PARTICIPATES** (0,N) PATIENT
  - PATIENT (1,1) **HAS** (1,1) RESERVATION
  - RESERVATION (0,N) **INVOLES** (0,N) MEDICAL EXAMINATION
  - MEDICAL EXAMINATION (1,1) **IS FOR** (0,N) REFERRAL
  - REFERRAL (0,N) **GETS** (0,N) PATIENT
  - PATIENT (1,1) **ISSUES** (0,N) REFERRAL

## Diagram relacji

**Transfers**(Id\_transfer, Start\_date, Start\_time, #FK\_user, #FK\_vehicle, #FK\_facility\_from, #FK\_facility\_to, Status)

Nazwa atrybutu	Znaczenie
Id_transfer	Identyfikator przejazdu w systemie
Start_date	Data wyjazdu
Start_time	Godzina wyjazdu
FK_user	Identyfikator kierowcy w systemie
FK_vehicle	Identyfikator pojazdu w systemie
FK_facility_from	Identyfikator placówki, z której odbywa się pojazd
FK_facility_to	Identyfikator placówki, do której odbywa się pojazd
Status	Status przejazdu Może przyjmować wartości Scheduled, Realised, Cancelled

**Transfers\_history**(Id\_transfer\_history, Id\_transfer, Start\_date, Start\_time, #FK\_user, #FK\_vehicle, #FK\_facility\_from, #FK\_facility\_to, Status)

Nazwa atrybutu	Znaczenie
Id_transfer_history	Unikatowy identyfikator stanu przejazdu
FK_transfer	Identyfikator przejazdu w systemie
Start_date	Data wyjazdu
Start_time	Godzina wyjazdu
FK_user	Identyfikator kierowcy w systemie
FK_vehicle	Identyfikator pojazdu w systemie
FK_facility_from	Identyfikator placówki, z której odbywa się pojazd
FK_facility_to	Identyfikator placówki, do której odbywa się pojazd
Status	Status przejazdu Może przyjmować wartości Scheduled, Realised, Cancelled
Create_date	Data utworzenia rekordu w tabeli

**Passengers**(Id\_passeneger, #FK\_patient, #FK\_transfer, Needs\_care, Status)

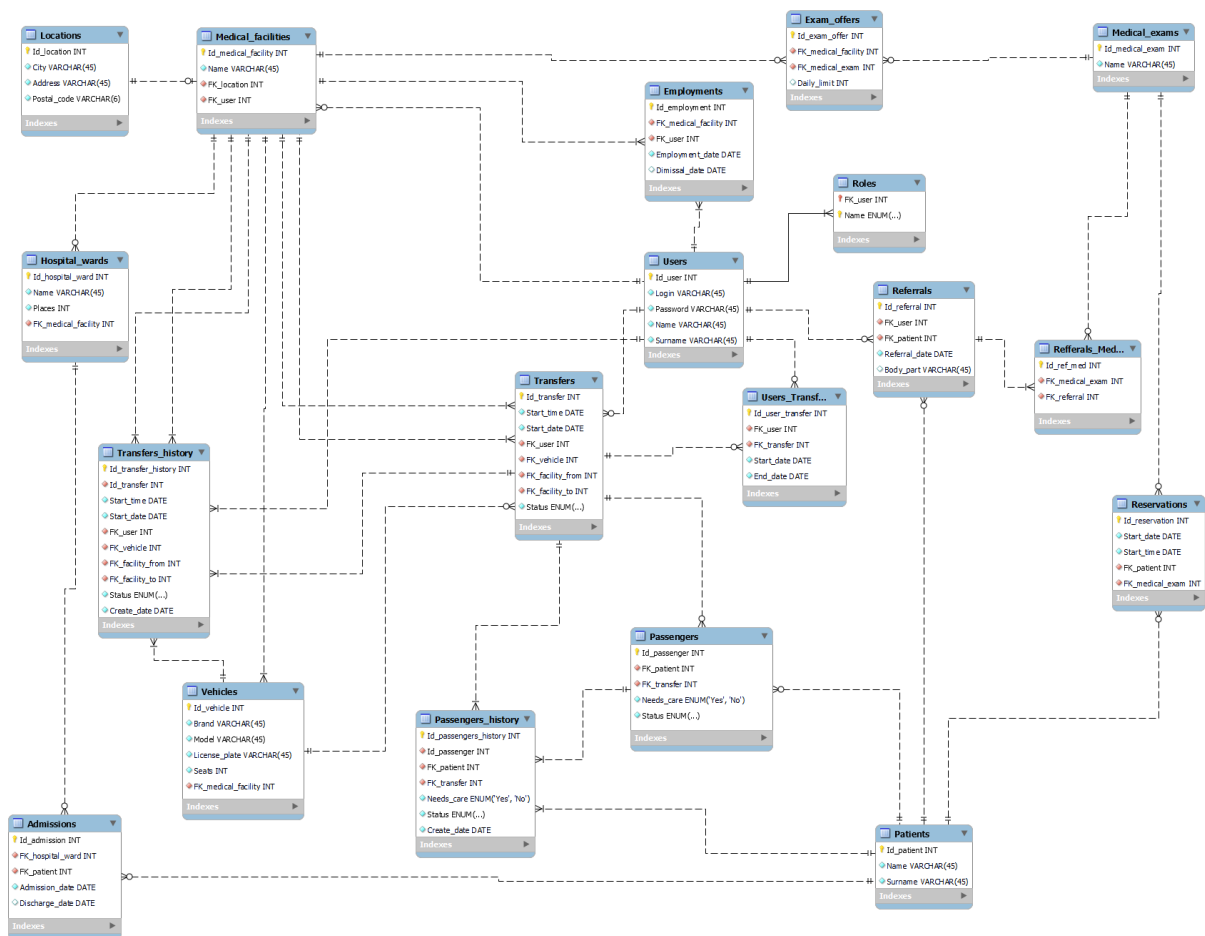
Nazwa atrybutu	Znaczenie
Id_passenger	Identyfikator rekordu w systemie
FK_patient	Identyfikator pacjenta w systemie
FK_transfer	Identyfikator przejazdu w systemie
Needs_care	Informacja czy pacjent wymaga dodatkowej opieki – wartość Tak/Nie
Status	Status pacjenta Może przyjmować wartości: Lying, Sitting, Walking

**Passengers\_history**(Id\_passeneger\_history, Id\_passenger, #FK\_patient, #FK\_transfer, Needs\_care, Status)

Nazwa atrybutu	Znaczenie
Id_passenger_history	Unikatowy identyfikator stanu pasażera
FK_passenger	Identyfikator rekordu w systemie
FK_patient	Identyfikator pacjenta w systemie
FK_transfer	Identyfikator przejazdu w systemie
Needs_care	Informacja czy pacjent wymaga dodatkowej opieki – wartość Tak/Nie
Status	Status pacjenta Może przyjmować wartości: Lying, Sitting, Walking
Create_date	Data utworzenia rekordu w tabeli

**Users\_Transfers**(Id\_User\_Transfer, #FK\_user, #FK\_transfer, Start\_date, End\_date, Is\_Active)

Nazwa atrybutu	Znaczenie
Id_user_transfer	Identyfikator rekordu w systemie
FK_user	Identyfikator medyka w systemie
FK_transfer	Identyfikator przejazdu w systemie
Start_date	Informacja, od kiedy przypisanie danego medyka do transportu jest aktualne
End_date	Informacja, do kiedy przypisanie danego medyka do transportu było aktualne



## Weryfikacja i aktualizacja więzów integralności oraz normalizacja

Ze względu na konieczność przechowywania danych historycznych dotyczących transportów, usuwanie ich – jak to było możliwe we wcześniejszej wersji bazy – traci sens. Nie będzie więc możliwe usuwanie danych z bazy, a co za tym idzie nie występuje usuwanie kaskadowe. Zamiast tego tabela Transport będzie przechowywać dodatkowy atrybut informujący o statusie: zaplanowany, wykonany, anulowany.

Więzy integralności tabel historycznych są analogiczne do tabel podstawowych. Więzy nie zostają naruszone przez zmiany wprowadzone w bazie.

Wprowadzane zmiany nie naruszają normalizacji naszej bazy danych do trzeciej postaci normalnej. Nie ma więc potrzeby przeprowadzania normalizacji dla zaktualizowanej bazy.

## **Etap 9.**

### **Funkcja ALTER**

Funkcja ALTER służy do modyfikowania już istniejącej bazy danych. Można za jej pomocą dodawać nowe kolumny, usuwać istniejące kolumny lub modyfikować istniejące kolumny. Niektóre systemy zarządzania bazą danych nie pozwalają usuwać istniejącej kolumny za pomocą funkcji ALTER – na szczęście, ta opcja jest dostępna w MySQL według dokumentacji.

Źródło:

[https://www.techonthenet.com/mysql/tables/alter\\_table.php](https://www.techonthenet.com/mysql/tables/alter_table.php)

Dodanie nowej kolumny do istniejącej tabeli:

```
ALTER TABLE table_name
  ADD new_column_name column_definition
  [ FIRST | AFTER column_name ];
```

Modyfikacja istniejącej kolumny:

```
ALTER TABLE table_name
  MODIFY column_name column_definition
  [ FIRST | AFTER column_name ];
```

Usunięcie kolumny:

```
ALTER TABLE table_name
  DROP COLUMN column_name;
```

### **Modyfikacja**

Zadana modyfikacja wymaga dodania kolumn do tabeli Transfers, Passengers i Users\_Transfers oraz stworzenia nowych tabel Transfers\_history i Passengers\_history. Wymaga również poprawienia skryptu w języku Python generującego dane.

## Skrypt SQL DDL

```
use medical_transport;
```

```
ALTER TABLE `medical_transport`.`Transfers`  
  ADD COLUMN `Status` ENUM("Scheduled", "Realised", "Cancelled") NOT NULL;
```

```
ALTER TABLE `medical_transport`.`Users_Transfers`  
  ADD COLUMN `Start_date` DATETIME NOT NULL,  
  ADD COLUMN `End_date` DATETIME;
```

```
CREATE TABLE IF NOT EXISTS `medical_transport`.`Transfers_history` (  
  `Id_transfer_history` INT NOT NULL AUTO_INCREMENT,  
  `FK_transfer` INT NOT NULL,  
  `Start_time` TIME NOT NULL,  
  `Start_date` DATE NOT NULL,  
  `FK_user` INT NOT NULL,  
  `FK_vehicle` INT NOT NULL,  
  `FK_facility_from` INT NOT NULL,  
  `FK_facility_to` INT NOT NULL,  
  `Status` ENUM("Scheduled", "Realised", "Cancelled") NOT NULL,  
  `Create_date` DATETIME default(current_timestamp()),  
  PRIMARY KEY (`Id_transfer_history`),  
  INDEX `FK_id_transfer_history` (`FK_user` ASC) VISIBLE,  
  INDEX `FK_user_idx` (`FK_user` ASC) VISIBLE,  
  INDEX `FK_vehicle_idx` (`FK_vehicle` ASC) VISIBLE,  
  INDEX `FK_facility_to_idx` (`FK_facility_to` ASC) VISIBLE,  
  CONSTRAINT  
    FOREIGN KEY (`FK_user`)  
    REFERENCES `medical_transport`.`Users` (`Id_user`),  
  CONSTRAINT  
    FOREIGN KEY (`FK_vehicle`)  
    REFERENCES `medical_transport`.`Vehicles` (`Id_vehicle`),  
  CONSTRAINT  
    FOREIGN KEY (`FK_facility_from`)  
    REFERENCES `medical_transport`.`Medical_facilities`  
    (`Id_medical_facility`),  
  CONSTRAINT  
    FOREIGN KEY (`FK_facility_to`)  
    REFERENCES `medical_transport`.`Medical_facilities`  
    (`Id_medical_facility`),  
  CONSTRAINT  
    FOREIGN KEY (`FK_transfer`)  
    REFERENCES `medical_transport`.`Transfers` (`Id_transfer`))  
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `medical_transport`.`Passengers_history` (  
  `Id_passenger_history` INT NOT NULL AUTO_INCREMENT,  
  `FK_passenger` INT NOT NULL,  
  `FK_patient` INT NOT NULL,  
  `FK_transfer` INT NOT NULL,  
  `Needs_care` ENUM('Yes', 'No') NOT NULL,
```



```

`Status` ENUM('Lying', 'Sitting', 'Walking') NOT NULL,
`Create_date` DATETIME default(current_timestamp()),
PRIMARY KEY (`Id_passenger_history`),
CONSTRAINT
    FOREIGN KEY (`FK_patient`)
    REFERENCES `medical_transport`.`Patients` (`Id_patient`),
CONSTRAINT
    FOREIGN KEY (`FK_transfer`)
    REFERENCES `medical_transport`.`Transfers` (`Id_transfer`),
CONSTRAINT
    FOREIGN KEY (`FK_passenger`)
    REFERENCES `medical_transport`.`Passengers` (`Id_passenger`))
ENGINE = InnoDB;

```

Aby dodać istniejące dane jako pierwszy wpis w tabeli historii używane są kwerendy:

```

insert into transfers_history (FK_transfer, Start_time, Start_date, FK_user,
FK_vehicle, FK_facility_from, FK_facility_to, `Status`, Create_date)
SELECT Id_transfer, Start_time, Start_date, FK_user, FK_vehicle,
FK_facility_from, FK_facility_to, `Status`, current_timestamp()
From medical_transport.transfers;

```

```

insert into passengers_history (FK_passenger, FK_patient, FK_transfer,
Needs_care, `Status`, Create_date)
SELECT Id_passenger, FK_patient, FK_transfer, Needs_care, `Status`,
current_timestamp()
From medical_transport.passengers;

```

```

UPDATE users_transfers set Start_date = current_timestamp();

```

## Skrypt w Pythonie

### Plik constants.py:

```

import datetime

START_DATE = datetime.date(2021, 1, 1)
END_DATE = datetime.date(2021, 11, 1)
EMPLOYMENT_DATE = datetime.date(1950, 1, 1)

SEATS = [1, 4, 5, 6, 7, 8, 9, 10, 13, 17, 18, 19, 20, 24]
ROLES = ["Adm", "Sup", "AEm", "Dri", "Doc", "Med"]
NEEDS_CARE = ["Yes", "No"]
STATUS = ["Lying", "Sitting", "Walking"]
TRANSFER_STATUS = ["Scheduled", "Realised", "Cancelled"]
BODY_PARTS = ["head", "spine", "arm", "hand", "finger", "leg", "knee",
"foot", "toe", "stomach", "shoulder", "eye",
"ear", "breast", "prostate", "heart", "lung", "hip", "pelvis",
"appendix", "bladder", "liver", "kidney",
"chest"]

```

```
HOSPITAL_WARDS = ["Oddzial Anestezjologii i Intensywnej Terapii", "Oddzial
Chirurgii Ogolnej", "Oddzial Dzieciacy",
                  "Oddzial Chirurgii Ogolnej i Onkologicznej", "Oddzial
Chirurgii Urazowo - Ortopedycznej",
                  "Oddzial Chorob Pluc", "Oddzial Chemioterapii", "Oddzial
Chorob Wewnetrznych", "Blok operacyjny",
                  "Oddzial Kardiologiczny", "Oddzial Nefrologiczny", "Oddzial
Chorob Wewnetrznych",
                  "Oddzial Neonatologiczny", "Oddzial Neurologiczny",
"Oddzial Pediatriczny", "Oddzial Nefrologii",
                  "Oddzial Polozniczo - Ginekologiczny", "Oddzial
Psychiatryczny", "Oddzial Endokrynologii",
                  "Oddzial Rehabilitacji Neurologicznej", "Oddzial
Rehabilitacji Kardiologicznej",
                  "Oddzial Rehabilitacyjny Ogolny", "Oddzial Udarowy",
"Oddzial Urologiczny", "Stacja Dializ",
                  "Szpitalny Oddzial Ratunkowy", "Oddzial Kardiochirurgii",
"Oddzial Obserwacyjno - Zakazny"
                  "Oddzial Dzienny Psychiatryczny", "Oddzial Chirurgii
Plastycznej", "Oddzial Neurochirurgiczny",
                  "Oddzial Okulistyczny", "Oddzial Psychiatryczny", "Oddzial
Leczenia Uzaleznien",
                  "Oddzial Rehabilitacyjny", "Oddzial Chirurgii Naczyniowej",
"Oddzial Nadciśnienia Tetniczego",
                  "Oddzial Gastroenterologii", "Oddzial Laryngologiczny",
"Oddzial Onkologii Klinicznej",
                  "Oddzial Reumatologiczny", "Oddzial Otolaryngologiczny",
"Oddzial Geriatryczny"]
```

```
USERS_LIMIT = 5000
LOCATIONS_LIMIT = 1250
MEDICAL_FACILITIES_LIMIT = 500
HOSPITAL_WARDS_LIMIT = 500
VEHICLES_LIMIT = 1000
TRANSFERS_LIMIT = 7000
PATIENTS_LIMIT = 7500
ADMISSIONS_LIMIT = 8500
REFERRALS_LIMIT = 9000
MEDICAL_EXAMS_LIMIT = 175
RESERVATIONS_LIMIT = 600
ROLES_LIMIT = 5500
EXAM_OFFERS_LIMIT = 2300
EMPLOYMENTS_LIMIT = 800
PASSENGERS_LIMIT = 8750
REFERRALS_MEDICAL_EXAMS = 3500
USERS_TRANSFERS_LIMIT = 4500
```

```
DAILY_LIMIT_LIMIT = 75
PLACES_IN_WARD_LIMIT = 80
POSTAL_CODE_LOWER_LIMIT = 10000
POSTAL_CODE_UPPER_LIMIT = 99999
```

## Plik tables.py:

```
from sqlalchemy import Column, Integer, String, Date, Time
from sqlalchemy import ForeignKey
from sqlalchemy.orm import relationship, declarative_base

Base = declarative_base()

class Users(Base):
    __tablename__ = 'Users'

    Id_user = Column(Integer, primary_key=True)
    Login = Column(String)
    Password = Column(String)
    Name = Column(String)
    Surname = Column(String)

    roles = relationship('Roles', backref='users')
    medical_facilities = relationship('Medical_facilities',
backref='users')
    transfers = relationship('Transfers', backref='users')
    referrals = relationship('Referrals', backref='users')
    employments = relationship('Employments', backref='users')
    users_transfers = relationship('Users_Transfers', backref='users')

class Locations(Base):
    __tablename__ = 'Locations'

    Id_location = Column(Integer, primary_key=True, autoincrement=True)
    City = Column(String)
    Address = Column(String)
    Postal_code = Column(String)

    medical_facilities = relationship('Medical_facilities',
backref='locations')

class Medical_facilities(Base):
    __tablename__ = 'Medical_facilities'

    Id_medical_facility = Column(Integer, primary_key=True,
autoincrement=True)
    Name = Column(String)
    FK_location = Column(Integer, ForeignKey('Locations.Id_location'))
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))

    hospital_wards = relationship('Hospital_wards',
backref='medical_facilities')
    employments = relationship('Employments', backref='medical_facilities')
    exam_offers = relationship('Exam_offers', backref='medical_facilities')
```

```

vehicles = relationship('Vehicles', backref='medical_facilities')

class Hospital_wards(Base):
    __tablename__ = 'Hospital_wards'

    Id_hospital_ward = Column(Integer, primary_key=True,
autoincrement=True)
    Name = Column(String)
    Places = Column(Integer)
    FK_medical_facility = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))

    admissions = relationship('Admissions', backref='hospital_wards')

class Vehicles(Base):
    __tablename__ = 'Vehicles'

    Id_vehicle = Column(Integer, primary_key=True, autoincrement=True)
    Brand = Column(String)
    Model = Column(String)
    License_plate = Column(String)
    Seats = Column(Integer)
    FK_medical_facility = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))

    transfers = relationship('Transfers', backref='vehicles')

class Transfers(Base):
    __tablename__ = 'Transfers'

    Id_transfer = Column(Integer, primary_key=True, autoincrement=True)
    Start_time = Column(Time)
    Start_date = Column(Date)
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))
    FK_vehicle = Column(Integer, ForeignKey('Vehicles.Id_vehicle'))
    FK_facility_from = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))
    FK_facility_to = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))
    Status = Column(String)

    passengers = relationship('Passengers', backref='transfers')
    users_transfers = relationship('Users_Transfers', backref='transfers')
    facility_to = relationship("Medical_facilities",
foreign_keys=[FK_facility_to])
    facility_from = relationship("Medical_facilities",
foreign_keys=[FK_facility_from])
    transfers_history = relationship("Transfers_history",
backref='transfers')

```

```
passengers_history = relationship('Passengers_history',  
backref='transfers')
```

```
class Roles(Base):  
    __tablename__ = 'Roles'  
  
    Id_role = Column(Integer, primary_key=True, autoincrement=True)  
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))  
    Name = Column(String)  
  
    unique_together = ('FK_user', 'Name')  
  
class Patients(Base):  
    __tablename__ = 'Patients'  
  
    Id_patient = Column(Integer, primary_key=True, autoincrement=True)  
    Name = Column(String)  
    Surname = Column(String)  
  
    reservations = relationship('Reservations', backref='patients')  
    admissions = relationship('Admissions', backref='patients')  
    referrals = relationship('Referrals', backref='patients')  
    passengers = relationship('Passengers', backref='patients')  
    passengers_history = relationship('Passengers_history',  
backref='patients')
```

```
class Admissions(Base):  
    __tablename__ = 'Admissions'  
  
    Id_admission = Column(Integer, primary_key=True, autoincrement=True)  
    FK_hospital_ward = Column(Integer,  
ForeignKey('Hospital_wards.Id_hospital_ward'))  
    FK_patient = Column(Integer, ForeignKey('Patients.Id_patient'))  
    Admission_date = Column(Date)  
    Discharge_date = Column(Date, nullable=True)
```

```
class Referrals(Base):  
    __tablename__ = 'Referrals'  
  
    Id_referral = Column(Integer, primary_key=True, autoincrement=True)  
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))  
    FK_patient = Column(Integer, ForeignKey('Patients.Id_patient'))  
    Referral_date = Column(Date)  
    Body_part = Column(String, nullable=True)  
  
    referrals_medical_exams = relationship('Referrals_Medical_exams',  
backref='referrals')
```

```
class Medical_exams(Base):
```

```

__tablename__ = 'Medical_exams'

Id_medical_exam = Column(Integer, primary_key=True, autoincrement=True)
Name = Column(String)

reservations = relationship('Reservations', backref='medical_exams')
exam_offers = relationship('Exam_offers', backref='medical_exams')
referrals_medical_exams = relationship('Referrals_Medical_exams',
backref='medical_exams')

class Reservations(Base):
    __tablename__ = 'Reservations'

    Id_reservaion = Column(Integer, primary_key=True, autoincrement=True)
    Start_date = Column(Date)
    Start_time = Column(Time)
    FK_patient = Column(Integer, ForeignKey('Patients.Id_patient'))
    FK_medical_exam = Column(Integer,
ForeignKey('Medical_exams.Id_medical_exam'))

class Exam_offers(Base):
    __tablename__ = 'Exam_offers'

    Id_exam_offers = Column(Integer, primary_key=True, autoincrement=True)
    FK_medical_facility = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))
    FK_medical_exam = Column(Integer,
ForeignKey('Medical_exams.Id_medical_exam'))
    Daily_limit = Column(Integer, nullable=True)

class Employments(Base):
    __tablename__ = 'Employments'

    Id_employment = Column(Integer, primary_key=True, autoincrement=True)
    FK_medical_facility = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))
    Employment_date = Column(Date)
    Dismissal_date = Column(Date, nullable=True)

class Passengers(Base):
    __tablename__ = 'Passengers'

    Id_passenger = Column(Integer, primary_key=True, autoincrement=True)
    FK_patient = Column(Integer, ForeignKey('Patients.Id_patient'))
    FK_transfer = Column(Integer, ForeignKey('Transfers.Id_transfer'))
    Needs_care = Column(String)
    Status = Column(String)

```

```

passengers_history = relationship('Passengers_history',
backref='passengers')

```

```

class Referrals_Medical_exams(Base):
    __tablename__ = 'Refferals_Medical_exams'

    Id_ref_med = Column(Integer, primary_key=True, autoincrement=True)
    FK_medical_exam = Column(Integer,
ForeignKey('Medical_exams.Id_medical_exam'))
    FK_referral = Column(Integer, ForeignKey('Referrals.Id_referral'))

```

```

class Users_Transfers(Base):
    __tablename__ = 'Users_Transfers'

    Id_user_transfer = Column(Integer, primary_key=True,
autoincrement=True)
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))
    FK_transfer = Column(Integer, ForeignKey('Transfers.Id_transfer'))
    Start_date = Column(Date)
    End_date = Column(Date, nullable=True)

```

```

class Transfers_history(Base):
    __tablename__ = 'Transfers_history'

    Id_transfer_history = Column(Integer, primary_key=True,
autoincrement=True)
    FK_transfer = Column(Integer, ForeignKey('Transfers.Id_transfer'))
    Start_time = Column(Time)
    Start_date = Column(Date)
    FK_user = Column(Integer, ForeignKey('Users.Id_user'))
    FK_vehicle = Column(Integer, ForeignKey('Vehicles.Id_vehicle'))
    FK_facility_from = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))
    FK_facility_to = Column(Integer,
ForeignKey('Medical_facilities.Id_medical_facility'))
    Status = Column(String)

    facility_to = relationship("Medical_facilities",
foreign_keys=[FK_facility_to])
    facility_from = relationship("Medical_facilities",
foreign_keys=[FK_facility_from])

```

```

class Passengers_history(Base):
    __tablename__ = 'Passengers_history'

    Id_passenger_history = Column(Integer, primary_key=True,
autoincrement=True)
    FK_passenger = Column(Integer, ForeignKey('Passengers.Id_passenger'))
    FK_patient = Column(Integer, ForeignKey('Patients.Id_patient'))

```

```

FK_transfer = Column(Integer, ForeignKey('Transfers.Id_transfer'))
Needs_care = Column(String)
Status = Column(String)
Create_date = Column(Date)

```

### **Plik main.py:**

```

from random import *
from datetime import datetime

from faker import Faker
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

from constants import *
from tables import *

fake = Faker()

user = 'root'
pwd = 'root123'
host = 'localhost'
db = 'medical_transport'
mysql_engine = create_engine('mysql://{0}:{1}@{2}/{3}'.format(user, pwd,
host, db))

def execute_query_1(session):
    result = session.execute(
        'SELECT users.Name, Surname, "Driver" As Rola FROM transfers left JOIN
users ON users.Id_user = transfers.FK_user '
        'left JOIN vehicles ON vehicles.Id_vehicle = transfers.FK_vehicle
WHERE transfers.Start_date '
        'BETWEEN :userDate1 AND :userDate2 AND vehicles.License_plate = :plate
UNION SELECT users.Name, Surname, '
        '"Medic" As Rola FROM medical_transport.users INNER JOIN
users_transfers ON users_transfers.FK_user = users.Id_user '
        'INNER JOIN transfers ON transfers.Id_transfer =
users_transfers.FK_transfer INNER JOIN vehicles '
        'ON vehicles.Id_vehicle = transfers.FK_vehicle WHERE
transfers.Start_date BETWEEN :userDate1 AND :userDate2 '
        'AND vehicles.License_plate = :plate UNION SELECT patients.Name,
patients.Surname, "Patient" As Rola '
        'FROM medical_transport.patients INNER JOIN passengers ON
passengers.FK_patient = patients.Id_patient '
        'INNER JOIN transfers ON transfers.Id_transfer =
passengers.FK_transfer INNER JOIN vehicles '
        'ON vehicles.Id_vehicle = transfers.FK_vehicle WHERE
transfers.Start_date BETWEEN :userDate1 AND :userDate2 '
        'AND vehicles.License_plate = :plate',
        {"userDate1": "2021-08-09", "userDate2": "2022-01-01", "plate": "8EE
G59"})

```



```

print('\nQuery 1:')
for r in result:
    print(r)

def execute_query_2(session):
    result = session.execute(
        'SELECT      medical_facilities.Name      FROM
medical_transport.medical_facilities INNER JOIN locations ON '
        'locations.Id_location = medical_facilities.FK_location INNER JOIN
exam_offers ON '
        'exam_offers.FK_medical_facility      =
medical_facilities.Id_medical_facility INNER JOIN medical_exams ON '
        'medical_exams.Id_medical_exam = exam_offers.FK_medical_exam WHERE
medical_exams.Name = :exam_name AND locations.City = :city'
        , {"exam_name": "OsaZyqqKWgizkHvNxMIqMpFdnQi", "city": "Eatonton"})

    print('\nQuery 2: ')
    for r in result:
        print(r)

def execute_query_3(session):
    result = session.execute(
        'SELECT      medical_facilities.Name,      hospital_wards.name,
hospital_wards.places, hospital_wards.places - COUNT(admissions.Id_admission)
AS free_places '
        'FROM      admissions      INNER      JOIN      hospital_wards      ON
hospital_wards.Id_hospital_ward = admissions.FK_hospital_ward '
        'INNER      JOIN      medical_facilities      ON
medical_facilities.Id_medical_facility = hospital_wards.FK_medical_facility '
        'INNER      JOIN      locations      ON      locations.Id_location      =
medical_facilities.FK_location '
        'WHERE isnull(admissions.Discharge_date) AND locations.city = :city
GROUP BY hospital_wards.Id_hospital_ward'
        , {"city": "Gaymouth"})

    print('\nQuery 3')
    for r in result:
        print(r)

def execute_query_4(session):
    result = session.execute(
        'SELECT      COUNT(referrals.Id_referral) FROM referrals INNER JOIN
refferals_medical_exams '
        'ON refferals_medical_exams.FK_referral = referrals.Id_referral INNER
JOIN medical_exams '
        'ON      medical_exams.Id_medical_exam      =
refferals_medical_exams.FK_medical_exam '
        'WHERE medical_exams.Name = :exam_name AND referrals.Referral_date
BETWEEN :date_start AND :date_end'

```

```
        , {"exam_name": "lSPPZuuXNxvfVGqVfdUVqrkGPdqUqsKc", "date_start":  
"2021-08-01", "date_end": "2022-01-01"})
```

```
print('\nQuery 4')  
for r in result:  
    print(r)
```

```
def execute_query_5(session):  
    result = session.execute(  
        'SELECT patients.name, patients.surname, COUNT(*) AS transports FROM  
patients LEFT JOIN passengers '  
        'ON passengers.FK_patient = patients.Id_patient GROUP BY  
patients.Id_patient ORDER BY transports DESC LIMIT 10')
```

```
print('\nQuery 5')  
for r in result:  
    print(r)
```

```
def execute_query_6(session):  
    result = session.execute(  
        'SELECT users.Name, users.Surname, employments.Employment_date,  
floor(datediff(now(), employments.Employment_date)/365) '  
        'AS seniority FROM users INNER JOIN employments ON employments.FK_user  
= users.Id_user '  
        'INNER JOIN medical_facilities ON  
medical_facilities.Id_medical_facility = employments.FK_medical_facility '  
        'WHERE Dismissal_date IS NULL AND medical_facilities.Name = :name  
ORDER BY seniority DESC'  
        , {"name": "ZoonhFzqgYRmeqJHmxjY"})
```

```
print('\nQuery 6')  
for r in result:  
    print(r)
```

```
def execute_query_7(session):  
    result = session.execute(  
        'SELECT AVG(transfers_number) AS Average_Transfers_For_Driver FROM  
(SELECT users.Name, users.Surname, COUNT(*) '  
        'AS transfers_number FROM users INNER JOIN roles ON roles.FK_user =  
users.Id_user INNER JOIN transfers '  
        'ON transfers.FK_user = users.Id_user GROUP BY users.Id_user ORDER BY  
transfers_number DESC) AS T')
```

```
print('\nQuery 7')  
for r in result:  
    print(r)
```

```
def execute_query_8(session):  
    result = session.execute(  

```

```

        'SELECT medical_facilities.Name, COUNT(transfers.Id_transfer) AS
Transfers from medical_facilities '
        'INNER JOIN transfers ON transfers.FK_facility_to =
medical_facilities.Id_medical_facility '
        'GROUP BY medical_facilities.Id_medical_facility ORDER BY Transfers
DESC LIMIT 10')

```

```

print('\nQuery 8')
for r in result:
    print(r)

```

```

def execute_query_9(session):
    result = session.execute(
        'SELECT medical_exams.Name as Exam, MAX(exam_offers.Daily_limit) AS
"Max Limit", medical_facilities.Name AS Facility '
        'FROM exam_offers LEFT JOIN medical_exams ON
medical_exams.Id_medical_exam = exam_offers.FK_medical_exam '
        'LEFT JOIN medical_facilities ON
medical_facilities.Id_medical_facility = exam_offers.FK_medical_facility '
        'GROUP BY medical_exams.Name LIMIT 10')

```

```

print('\nQuery 9')
for r in result:
    print(r)

```

```

def execute_query_10(session):
    result = session.execute(
        'SELECT DISTINCT hospital_wards.Name FROM hospital_wards LEFT JOIN
medical_facilities '
        'ON medical_facilities.Id_medical_facility =
hospital_wards.FK_medical_facility LEFT JOIN locations '
        'ON locations.Id_location = medical_facilities.FK_location WHERE
locations.City = :city'
        , {"city": "East Robertomouth"})

```

```

print('\nQuery 10')
for r in result:
    print(r)

```

```

def execute_query_11(session):
    result = session.execute(
        'SELECT FK_transfer, Start_date, Start_time, users.Name AS Driver,
vehicles.License_plate, medical_facilities.Name AS "Facility to", "Status",
Create_date '
        'FROM transfers_history INNER JOIN medical_transport.users ON
users.Id_user = transfers_history.FK_user '
        'INNER JOIN medical_transport.vehicles ON vehicles.Id_vehicle =
transfers_history.FK_vehicle '
        'INNER JOIN medical_transport.medical_facilities ON
medical_facilities.Id_medical_facility = transfers_history.FK_facility_to '

```

```

        'WHERE transfers_history.FK_transfer = 6')

print('\nQuery 11')
for r in result:
    print(r)

def execute_queries(session):
    execute_query_1(session)
    execute_query_2(session)
    execute_query_3(session)
    execute_query_4(session)
    execute_query_5(session)
    execute_query_6(session)
    execute_query_7(session)
    execute_query_8(session)
    execute_query_9(session)
    execute_query_10(session)

def get_logins(session):
    return [user.Login for user in session.query(Users).all()]

def get_supervisors(session):
    return [medical_facility.FK_user for medical_facility in
session.query(Medical_facilities).all()]

def get_license_plates(session):
    return [vehicle.License_plate for vehicle in
session.query(Vehicles).all()]

def get_hospital_wards(session):
    return [(hospital_ward.Name, hospital_ward.FK_medical_facility)
for hospital_ward in session.query(Hospital_wards).all()]

def get_users_with_role(session, role_id):
    users = session.query(Roles).all()
    users_with_role = []
    for user in users:
        if user.Name == ROLES[role_id]:
            users_with_role.append(user.FK_user)
    return users_with_role

def generate_users(session):
    logins = get_logins(session)
    for i in range(0, USERS_LIMIT):
        login = fake.user_name()
        while logins.count(login) != 0:

```

```

        login = fake.user_name()
        logins.append(login)
        new_user = Users(
            Login=login,
            Password=fake.password(),
            Name=fake.first_name(),
            Surname=fake.last_name()
        )
        session.add(new_user)
    session.commit()

```

```

def generate_roles(session):
    user_index = 1
    for i in range(0, MEDICAL_FACILITIES_LIMIT):
        role = Roles(
            FK_user=user_index,
            Name=ROLES[1]
        )
        user_index = user_index + 1
        session.add(role)
        for i in range(0, min(USERS_LIMIT - MEDICAL_FACILITIES_LIMIT,
ROLES_LIMIT)):
            role = Roles(
                FK_user=user_index,
                Name=choice(ROLES)
            )
            user_index = user_index + 1
            session.add(role)
    for i in range(0, ROLES_LIMIT - USERS_LIMIT):
        role = Roles(
            FK_user=randint(1, USERS_LIMIT),
            Name=choice(ROLES)
        )
        session.add(role)
    session.commit()

```

```

def generate_locations(session):
    for i in range(0, LOCATIONS_LIMIT):
        location = Locations(
            City=fake.city(),
            Address=fake.street_address(),
            Postal_code=randint(POSTAL_CODE_LOWER_LIMIT,
POSTAL_CODE_UPPER_LIMIT)
        )
        session.add(location)
    session.commit()

```

```

def generate_medical_facilities(session):
    supervisors = get_supervisors(session)
    possible_supervisors = get_users_with_role(session, 1)

```

```

for i in range(0, MEDICAL_FACILITIES_LIMIT):
    supervisor = choice(possible_supervisors)
    while supervisors.count(supervisor) != 0:
        supervisor = choice(possible_supervisors)
    supervisors.append(supervisor)
    medical_facility = Medical_facilities(
        Name=fake.pystr(),
        FK_location=randint(1, MEDICAL_FACILITIES_LIMIT),
        FK_user=supervisor
    )
    session.add(medical_facility)
session.commit()

def generate_hospital_wards(session):
    hospital_wards = []
    for i in range(0, HOSPITAL_WARDS_LIMIT):
        medical_facility = randint(1, MEDICAL_FACILITIES_LIMIT)
        hospital_ward = choice(HOSPITAL_WARDS)
        while hospital_wards.count([hospital_ward, medical_facility]) != 0:
            hospital_ward = choice(HOSPITAL_WARDS)
        hospital_ward_A = Hospital_wards(
            Name=hospital_ward,
            Places=randint(0, PLACES_IN_WARD_LIMIT),
            FK_medical_facility=medical_facility
        )
        hospital_wards.append([hospital_ward, medical_facility])
        session.add(hospital_ward_A)
    session.commit()

def generate_vehicles(session):
    license_plates = get_license_plates(session)
    for i in range(0, VEHICLES_LIMIT):
        license_plate = fake.license_plate()
        while license_plates.count(license_plate) != 0:
            license_plate = fake.license_plate()
        license_plates.append(license_plate)
        seats = int(choice(SEATS))
        vehicle = Vehicles(
            Brand=fake.pystr(2, 20),
            Model=fake.pystr(3, 20),
            License_plate=license_plate,
            Seats=seats,
            FK_medical_facility=randint(1, MEDICAL_FACILITIES_LIMIT)
        )
        session.add(vehicle)
    session.commit()

def generate_transfers(session):
    possible_drivers = get_users_with_role(session, 3)
    for i in range(0, TRANSFERS_LIMIT):

```

```

start_time = fake.time()
start_date = fake.date_between(start_date=START_DATE, end_date='+1y')
fk_user = choice(possible_drivers)
fk_vehicle = randint(1, VEHICLES_LIMIT)
fk_facility_to = randint(1, MEDICAL_FACILITIES_LIMIT)
fk_facility_from = randint(1, MEDICAL_FACILITIES_LIMIT)
status = choice(TRANSFER_STATUS)
transfer = Transfers(
    Start_time=start_time,
    Start_date=start_date,
    FK_user=fk_user,
    FK_vehicle=fk_vehicle,
    FK_facility_to=fk_facility_to,
    FK_facility_from=fk_facility_from,
    Status=status
)
transfer_history = Transfers_history(
    FK_transfer=i+1,
    Start_time=start_time,
    Start_date=start_date,
    FK_user=fk_user,
    FK_vehicle=fk_vehicle,
    FK_facility_from=fk_facility_from,
    FK_facility_to=fk_facility_to,
    Status=status,
    Create_date=datetime.datetime.now()
)
session.add(transfer)
session.add(transfer_history)
session.commit()

```

```

def generate_patients(session):
    for i in range(0, PATIENTS_LIMIT):
        patient = Patients(
            Name=fake.first_name(),
            Surname=fake.last_name()
        )
        session.add(patient)
    session.commit()

```

```

def generate_admissions(session):
    for i in range(0, ADMISSIONS_LIMIT):
        admission_date = fake.date_between(START_DATE, END_DATE)
        discharge_date = None
        if fake.pybool():
            discharge_date = fake.date_between(admission_date, END_DATE)
        admission = Admissions(
            Admission_date=admission_date,
            Discharge_date=discharge_date,
            FK_hospital_ward=randint(1, HOSPITAL_WARDS_LIMIT),
            FK_patient=randint(1, PATIENTS_LIMIT))

```

```

        session.add(admission)
    session.commit()

def generate_referrals(session):
    possible_doctors = get_users_with_role(session, 4)
    for i in range(0, REFERRALS_LIMIT):
        body_parts = None
        if fake.pybool():
            body_parts = choice(BODY_PARTS)
        referrals = Referrals(
            Referral_date=fake.date_between(START_DATE, END_DATE),
            Body_part=body_parts,
            FK_user=choice(possible_doctors),
            FK_patient=randint(1, PATIENTS_LIMIT)
        )
        session.add(referrals)
    session.commit()

def generate_medical_exams(session):
    for i in range(0, MEDICAL_EXAMS_LIMIT):
        medical_exam = Medical_exams(
            Name=fake.pystr(5, 45),
        )
        session.add(medical_exam)
    session.commit()

def generate_reservations(session):
    for i in range(0, RESERVATIONS_LIMIT):
        reservation = Reservations(
            Start_date=fake.date_between(start_date=START_DATE,
end_date='+1y'),
            Start_time=fake.time(),
            FK_patient=randint(1, PATIENTS_LIMIT),
            FK_medical_exam=randint(1, MEDICAL_EXAMS_LIMIT)
        )
        session.add(reservation)
    session.commit()

def generate_exam_offers(session):
    for i in range(0, RESERVATIONS_LIMIT):
        if fake.pybool():
            daily_limit = randint(1, DAILY_LIMIT_LIMIT)
        else:
            daily_limit = None
        exam_offers = Exam_offers(
            FK_medical_facility=randint(1, MEDICAL_FACILITIES_LIMIT),
            FK_medical_exam=randint(1, MEDICAL_EXAMS_LIMIT),
            Daily_limit=daily_limit
        )

```



```
        session.add(exam_offers)
    session.commit()
```

```
def generate_employments(session):
    possible_doctors = get_users_with_role(session, 4)
    for i in range(0, MEDICAL_EXAMS_LIMIT):
        employment_date = fake.date_between(EMPLOYMENT_DATE, END_DATE)
        dismissal_date = None
        if fake.pybool():
            dismissal_date = fake.date_between(employment_date, END_DATE)
        employment = Employments(
            FK_medical_facility=randint(1, MEDICAL_FACILITIES_LIMIT),
            FK_user=choice(possible_doctors),
            Employment_date=employment_date,
            Dismissal_date=dismissal_date
        )
        session.add(employment)
    session.commit()
```

```
def generate_passengers(session):
    for i in range(0, PASSENGERS_LIMIT):
        fk_patient = randint(1, PATIENTS_LIMIT)
        fk_transfer = randint(1, TRANSFERS_LIMIT)
        needs_care = NEEDS_CARE[randint(0, 1)]
        status = choice(STATUS)
        passenger = Passengers(
            FK_patient=fk_patient,
            FK_transfer=fk_transfer,
            Needs_care=needs_care,
            Status=status
        )
        passenger_history = Passengers_history(
            FK_passenger=i+1,
            FK_patient=fk_patient,
            FK_transfer=fk_transfer,
            Needs_care=needs_care,
            Status=status,
            Create_date=datetime.datetime.now()
        )
        session.add(passenger)
        session.add(passenger_history)
    session.commit()
```

```
def generate_referrals_medical_exams(session):
    for i in range(0, REFERRALS_MEDICAL_EXAMS):
        referrals_medical_exams = Referrals_Medical_exams(
            FK_medical_exam=randint(1, MEDICAL_EXAMS_LIMIT),
            FK_referral=randint(1, REFERRALS_LIMIT),
        )
        session.add(referrals_medical_exams)
```

```
session.commit()
```

```
def generate_users_transfers(session):
    possible_medics = get_users_with_role(session, 5)
    facility_from = randint(1, MEDICAL_FACILITIES_LIMIT)
    facility_to = randint(1, MEDICAL_FACILITIES_LIMIT)
    while facility_to == facility_from:
        facility_to = randint(1, MEDICAL_FACILITIES_LIMIT)
    for i in range(0, USERS_TRANSFERS_LIMIT):
        users_transfers = Users_Transfers(
            FK_user=choice(possible_medics),
            FK_transfer=randint(1, TRANSFERS_LIMIT),
            Start_date=datetime.datetime.now(),
            End_date=None
        )
        session.add(users_transfers)
    session.commit()
```

```
def generate_all_silent(session):
    generate_users(session)
    generate_roles(session)
    generate_locations(session)
    generate_medical_facilities(session)
    generate_hospital_wards(session)
    generate_vehicles(session)
    generate_transfers(session)
    generate_patients(session)
    generate_admissions(session)
    generate_referrals(session)
    generate_medical_exams(session)
    generate_reservations(session)
    generate_exam_offers(session)
    generate_employments(session)
    generate_passengers(session)
    generate_referrals_medical_exams(session)
    generate_users_transfers(session)
```

```
def generate_all_verbose(session):
    generate_users(session)
    print("Generating Users finished")
    generate_roles(session)
    print("Generating Roles finished")
    generate_locations(session)
    print("Generating Locations finished")
    generate_medical_facilities(session)
    print("Generating Medical_facilities finished")
    generate_hospital_wards(session)
    print("Generating Hospital_wards finished")
    generate_vehicles(session)
    print("Generating Vehicles finished")
```

```

generate_transfers(session)
print("Generating Transfers finished")
generate_patients(session)
print("Generating Patients finished")
generate_admissions(session)
print("Generating Admissions finished")
generate_referrals(session)
print("Generating Referrals finished")
generate_medical_exams(session)
print("Generating Medical_exams finished")
generate_reservations(session)
print("Generating Reservations finished")
generate_exam_offers(session)
print("Generating Exam_offers finished")
generate_employments(session)
print("Generating Employments finished")
generate_passengers(session)
print("Generating Passengers finished")
generate_referrals_medical_exams(session)
print("Generating Referrals_Medical_exams finished")
generate_users_transfers(session)
print("Generating Users_Transfers finished")
print()
print("-----Database generated :)")
print()

```

```

if __name__ == '__main__':
    Session = sessionmaker(bind=mysql_engine, autoflush=False)
    current_session = Session()

    # generate_all_silent(current_session)
    generate_all_verbose(current_session)

    # execute_queries(current_session)
    execute_query_11(current_session)

    current_session.close()

```

## Kwerendy:

Kwerenda testująca poprawność działania modyfikacji:

```
Select      FK_transfer,      Start_date,      Start_time,      users.Name,
vehicles.License_plate, medical_facilities.Name as `Facility to`, `Status`,
Create_date
from transfers_history
inner      join      medical_transport.users      on      users.Id_user      =
transfers_history.FK_user
inner      join      medical_transport.vehicles      on      vehicles.Id_vehicle      =
transfers_history.FK_vehicle
inner      join      medical_transport.medical_facilities      on
medical_facilities.Id_medical_facility = transfers_history.FK_facility_to
where Id_transfer = '&id';
```

Zadana modyfikacja nie wymaga poprawiania poprzednich kwerend.

Poprawność i spójność danych została zachowana dzięki odpowiednim kwerendom.

# End of Part 1