# Dokumentacja bazy MPK

## Spis treści

* [Dokumentacja bazy MPK](#dokumentacja-bazy-mpk)
  + [Spis treści](#spis-treści)
  + [Założenia projektowe i opis](#założenia-projektowe-i-opis)
  + [Diagram](#diagram)

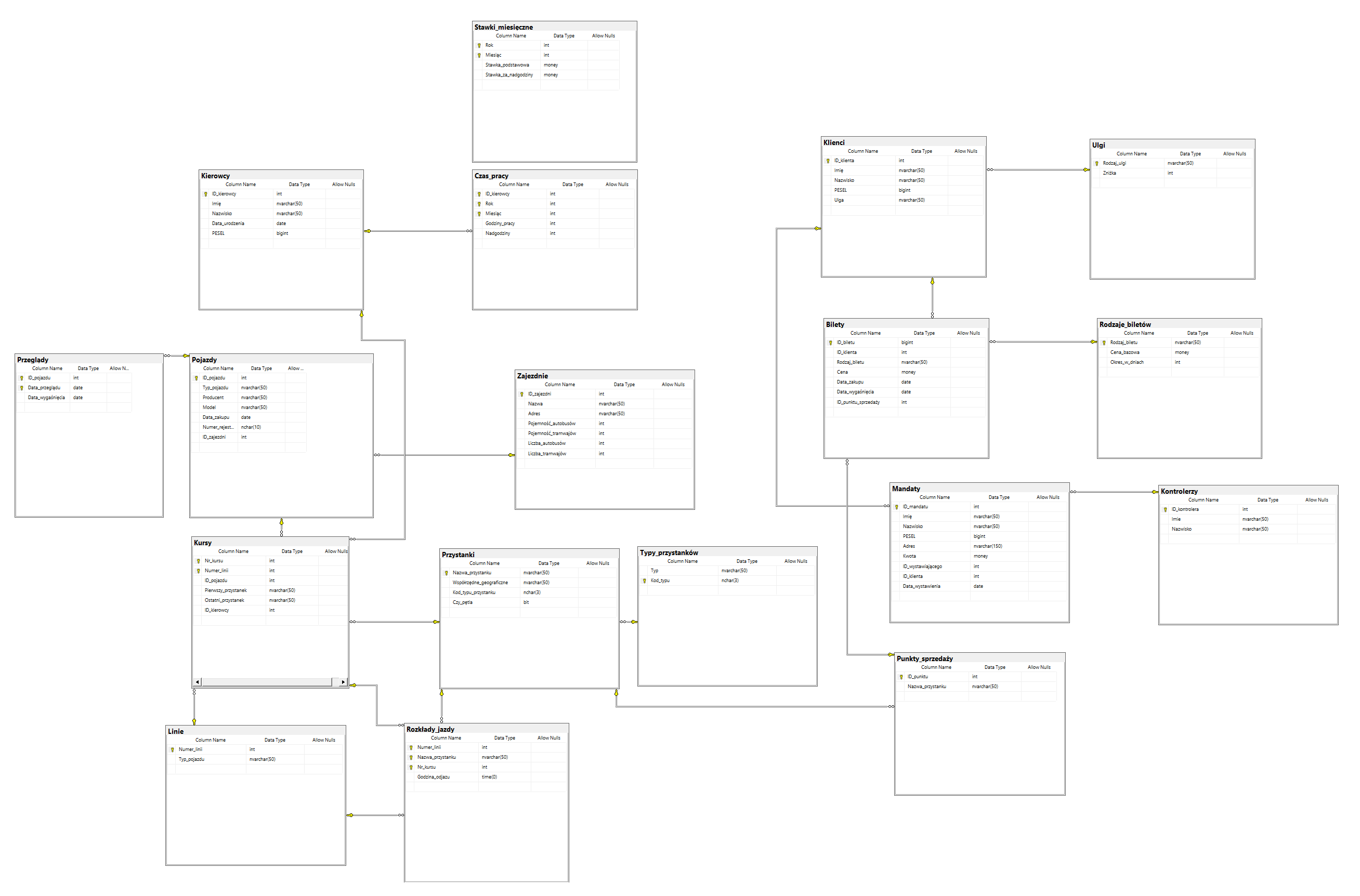
## Założenia projektowe i opis

Baza danych MPK zawiera informacje o działaniu małego przedsiębiorstwa komunikacyjnego. Przechowuje informacje o pojazdach, przystankach, kierowcach i pasażerach, oraz powiązane z tymi zagadnieniami informacje. Największy nacisk położono na informacje o technicznej stronie funkcjonowania przedsiębiorstwa, to jest dokładnych informacjach o istniejących liniach, ich rozkładach jazdy, a także pojazdach oraz ich stanie technicznym.

Przyjęte założenia:

* Czas pracy kierowcy podzielony jest na czas główny i nadgodziny, kierowcy są rozliczani w trybie miesięcznym
* Pojazdy zawsze muszą być przypisane do zajezdni (lub specjalnego typu zajezdni Magazyn)
* Zajezdnie mogą być typu autobusowego, tramwajowego lub mieszanego
* Możliwe typy pojazdów to autobus i tramwaj
* Jeden pojazd może być przypisany do wielu kursów
* Jeden kierowca może być przypisany do wielu kursów
* Może istnieć tylko jedna linia o podanym numerze, w obrębie linii może istnieć wiele kursów o unikatowych numerach identyfikacyjnych
* Kursy w obrębie linii mogą różnić się rozkładami jazdy
* Rozkład jazdy składa się z przystanków, oraz godzin odjazdu, kolejność rozkładu ustalają rosnące godziny odjazdów
* Przystanki są identyfikowane tylko swoją nazwą, ich lokalizacja jest określana na podstawie współrzędnych geograficznych
* Możliwe typy przystanków to autobusowy, tramwajowy i autobusowo-tramwajowy
* Przechowywane są informacje tylko o klientach posiadających bilety okresowe
* Klient może posiadać jedną ulgę, w przypadku różnych ulg wybierana jest najwyższa
* Przechowywana jest cała historia biletów okresowych, ich aktywność ustala się na podstawie daty wygaśnięcia
* Cena biletów pobierana jest z osobnej tabeli, **może zmieniać się w czasie**
* Bilety sprzedawane są w punktach sprzedaży, które są zlokalizowane tylko na przystankach
* Przechowywane są tylko informacje o nieuregulowanych mandatach, zapłacone są usuwane
* Mandat może być wystawiony przez kontrolera istniejącego w bazie danych przedsiębiorstwa lub przez zewnętrznych kontrolerów
* Numer identyfikacyjny mandatu ustalany jest przez wystawiającego
* Klient z niezapłaconym mandatem nie może zakupić biletu

## Diagram



diagram

# # Historia\_biletow\_osoby

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| Identyfikator klienta | INT |

## Wyjście

|  |  |
| --- | --- |
| Nazwa | Typ |
| Historia\_biletow\_osoby | TABLE |

## Opis działania

Wypisz wszystkie bilety zakupione przez klienta o podanym ID. Otrzymamy informację o rodzaju biletu, cenie, dacie zakupu, dacie wygaśnięcia biletu, punkcie sprzedaży biletu oraz wszystkie dane klienta w tym przysługujące mu w danym momencie ulgi.

## Kod źródłowy

GO  
CREATE FUNCTION Historia\_biletow\_osoby (@klient AS INT)  
RETURNS TABLE  
AS  
RETURN  
 SELECT ID\_biletu,Rodzaj\_biletu,Cena,Data\_zakupu,Data\_wygaśnięcia,ID\_punktu\_sprzedaży,  
 K.ID\_klienta,Imię,Nazwisko,PESEL,Ulga FROM Bilety as B  
 JOIN Klienci as K on B.ID\_klienta = K.ID\_klienta  
 WHERE K.ID\_klienta = @klient  
GO

[link do kodu](../../functions/Historia_biletow_osoby.sql)

# # Rozklad\_jazdy\_kierowcy

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| Identyfikator kierowcy | INT |

## Wyjście

|  |  |
| --- | --- |
| Nazwa | Typ |
| Rozklad\_jazdy\_kierowcy | TABLE |

## Opis działania

Wypisz wszystkie kursy kierowcy o podanym id. Dostajemy pełne informacje o kierowcy, przystanek początkowy, końcowy kursu oraz przystanki wraz z godzinami odjazdu.

## Kod źródłowy

GO  
CREATE FUNCTION Rozklad\_jazdy\_kierowcy (@Id\_k AS INT)  
RETURNS TABLE  
AS  
RETURN  
 SELECT K.ID\_kierowcy,K.Imię,K.Nazwisko,K.Data\_urodzenia,K.PESEL,Ku.Pierwszy\_przystanek,Ku.Ostatni\_przystanek,  
 R.\* FROM Kierowcy as K  
 LEFT JOIN Kursy as Ku on K.ID\_kierowcy = Ku.ID\_kierowcy  
 LEFT JOIN Rozkłady\_jazdy as R on Ku.Nr\_kursu = R.Nr\_kursu  
 WHERE K.ID\_kierowcy = @Id\_k  
GO

[link do kodu](../../functions/Rozklad_jazdy_kierowcy.sql)

# Sprzedaż\_w\_punkcie

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| ID\_punktu | INT |

## Wyjście

|  |  |
| --- | --- |
| Nazwa | Typ |
| Sprzedaż\_w\_punkcie | TABLE |

## Opis działania

Wybierz z tabeli Bilety sumę zysku ze sprzedanych biletów sprzedanych w podanym punkcie (ID\_punktu), pogrupowane wedługo lat i miesięcy.

## Kod źródłowy

CREATE OR ALTER FUNCTION Sprzedaż\_w\_punkcie (@ID\_punktu INT)  
RETURNS TABLE  
AS  
RETURN  
(  
 SELECT YEAR(B.Data\_zakupu) AS Rok, MONTH(B.Data\_zakupu) AS Miesiąc, SUM(B.Cena) AS Zysk  
 FROM Bilety AS B  
 GROUP BY YEAR(B.Data\_zakupu), MONTH(B.Data\_zakupu)  
);

[link do kodu](../../functions/Sprzedaż_w_punkcie.sql)

# # Trasa\_linii\_pojazdu

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| Numer\_linii | INT |

## Wyjście

|  |  |
| --- | --- |
| Nazwa | Typ |
| Trasa\_linii\_pojazdu | TABLE |

## Opis działania

Wypisz wszystkie kursy pojazdu o podanym numerze linii, przystanki przez które przejeżdża oraz godziny odjazdu.

## Kod źródłowy

GO  
CREATE FUNCTION Trasa\_linii\_pojazdu (@Num\_l AS INT)  
RETURNS TABLE  
AS  
RETURN  
 SELECT L.Numer\_linii,Typ\_pojazdu,Nr\_kursu,Nazwa\_przystanku, Godzina\_odjazu FROM Linie as L  
 LEFT JOIN Rozkłady\_jazdy as R on L.Numer\_linii = R.Numer\_linii  
 WHERE L.Numer\_linii = @Num\_l  
GO

[link do kodu](../../functions/Trasa_linii_pojazdu.sql)

# Wygasające\_przeglądy

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| Rok | INT |
| Miesiąc | INT |
| Dzień | INT |

## Wyjście

|  |  |
| --- | --- |
| Nazwa | Typ |
| Wygasające\_przeglądy | TABLE |

## Opis działania

Zwróć tabelę zawierającą wszystkie pojazdy, których przeglądy wygasną przed datą podaną w parametrach wejściowych.

## Kod źródłowy

CREATE OR ALTER FUNCTION Wygasające\_przeglądy (@Rok INT, @Miesiąc INT = 1, @Dzień INT = 1)  
RETURNS @Wygasające\_przeglądy TABLE  
(  
 ID\_pojazdu INT PRIMARY KEY NOT NULL,  
 Data\_wygaśnięcia DATE NOT NULL  
)  
AS  
BEGIN  
 DECLARE @Data DATE  
 SET @Data = CAST(CAST(@Miesiąc AS varchar) + '-' + CAST(@Dzień AS varchar) + '-' + CAST(@Rok AS varchar) AS date)  
   
 INSERT INTO @Wygasające\_przeglądy(ID\_pojazdu, Data\_wygaśnięcia)  
 SELECT P.ID\_pojazdu, P.Data\_wygaśnięcia  
 FROM Przeglądy AS P  
 WHERE P.Data\_wygaśnięcia < @Data  
 RETURN  
END

[link do kodu](../../functions/Wygasające_przeglądy.sql)

# Znajdź\_trasę

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| Przystanek\_początkowy | NVARCHAR(50) |
| Przystanek\_docelowy | NVARCHAR(50) |
| Godzina\_odjazdu | TIME(0) |

## Wyjście

|  |  |
| --- | --- |
| Nazwa | Typ |
| - | TABLE |

## Opis działania

Znajdź trasę pomiędzy podanymi przystankami (Przystanek\_początkowy i Przystanek\_docelowy) i odjeżdżających najwcześniej o podanej godzinie (Godzina\_odjazdu). Zwraca maksymalnie 5 kursów, posortowanych według godziny odjazdu.

## Kod źródłowy

CREATE OR ALTER FUNCTION Znajdź\_trasę   
(  
 @Przystanek\_początkowy NVARCHAR(50),   
 @Przystanek\_docelowy NVARCHAR(50),   
 @Godzina\_odjazdu TIME(0)  
)  
RETURNS TABLE  
AS  
RETURN  
(  
 SELECT TOP 5 Numer\_linii, Godzina\_odjazu  
 FROM Rozkłady\_jazdy R1  
 WHERE (Nazwa\_przystanku LIKE @Przystanek\_początkowy) AND EXISTS   
 (  
 SELECT \*  
 FROM Rozkłady\_jazdy R2  
 WHERE (Nazwa\_przystanku LIKE @Przystanek\_docelowy) AND (R1.Numer\_linii = R2.Numer\_linii) AND (R2.Godzina\_odjazu >= R1.Godzina\_odjazu)  
 ) AND Godzina\_odjazu >= @Godzina\_odjazdu  
 ORDER BY Godzina\_odjazu  
);  
GO

[link do kodu](../../functions/Znajdź_trasę.sql)

# Dodaj\_bilet

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| Id\_klienta | INT |
| Rodzaj\_biletu | nvarchar(50) |
| Data\_zakupu | date |
| Id\_punktu\_sprzedaży | INT |

## Opis działania

Sprawdź czy deklarowany punkt istnieje, sprawdź czy klient istnieje, sprawdź czy nie próbujesz dodać biletu osobie, która ma już ważny bilet. Dodaj bilet do tabeli Bilety.

## Kod źródłowy

CREATE OR ALTER PROCEDURE Dodaj\_bilet  
 @Id\_klienta INT,  
 @Rodzaj\_biletu nvarchar(50),  
 @Data\_zakupu date,  
 @Id\_punktu\_sprzedaży INT  
AS  
 IF NOT EXISTS (  
 SELECT \*   
 FROM Punkty\_sprzedaży  
 WHERE ID\_punktu = @Id\_punktu\_sprzedaży)  
 BEGIN  
 RAISERROR('Nie ma takiego punktu sprzedaży',16,1)  
 RETURN;  
 END  
 ELSE IF NOT EXISTS (  
 SELECT \*   
 FROM Klienci  
 WHERE ID\_klienta = @Id\_klienta)  
 BEGIN  
 RAISERROR('Nie ma takiego klienta',16,2)  
 RETURN;  
 END  
 ELSE IF EXISTS (  
 SELECT \* FROM Bilety  
 WHERE ID\_klienta = @Id\_klienta AND Data\_wygaśnięcia>@Data\_zakupu)  
 BEGIN  
 RAISERROR('Twój bilet jest już ważny',16,2)  
 RETURN;  
 END  
 ELSE  
 BEGIN  
 INSERT INTO Bilety  
 VALUES(@Id\_klienta,@Rodzaj\_biletu,-1,@Data\_zakupu,@Data\_zakupu,@Id\_punktu\_sprzedaży)  
 END

[link do kodu](../../procs/Dodaj_bilet.sql)

# Dodaj\_kurs

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| Nr\_kursu | INT |
| Nr\_linii | INT |
| ID\_pojazdu | INT |
| ID\_kierowcy | INT |
| Rozkład | Przystanek\_w\_rozkładzie |

## Opis działania

Sprawdź czy dane są poprawne, sprawdź czy kolejne przystanki podane w rozkładzie mają coraz późniejsze godziny odjazdu. Dodaj kurs do tabeli Kursy oraz rozkład jazdy do tabeli Rozkłady\_jazdy.

## Kod źródłowy

CREATE OR ALTER PROCEDURE Dodaj\_kurs  
 @Nr\_kursu INT,  
 @Nr\_linii INT,  
 @ID\_pojazdu INT,  
 @ID\_kierowcy INT,  
 @Rozkład [Przystanek\_w\_rozkładzie] READONLY  
AS  
 SET NOCOUNT ON  
 IF NOT EXISTS (  
 SELECT L.Numer\_linii  
 FROM Linie AS L  
 WHERE L.Numer\_linii = @Nr\_linii  
 )  
 BEGIN  
 RAISERROR ('Taka linia nie istnieje', 16, 1)  
 RETURN;  
 END  
  
 IF EXISTS (  
 SELECT K.Nr\_kursu  
 FROM Kursy AS K  
 WHERE (K.Nr\_kursu = @Nr\_kursu) AND (K.Numer\_linii = @Nr\_linii)  
 )  
 BEGIN  
 RAISERROR ('Kurs o podanym numerze już istnieje', 16, 2)  
 RETURN;  
 END  
  
 IF EXISTS (  
 SELECT \*  
 FROM @Rozkład AS L  
 CROSS JOIN @Rozkład AS R  
 WHERE (L.Numer\_przystanku < R.Numer\_przystanku) AND (L.Godzina\_odjazdu > R.Godzina\_odjazdu)  
 )  
 BEGIN  
 RAISERROR ('Nieprawidłowa kolejność przystanków', 16, 3)  
 RETURN;  
 END  
  
 IF NOT EXISTS (  
 SELECT P.ID\_pojazdu  
 FROM Pojazdy AS P  
 WHERE P.ID\_pojazdu = @ID\_pojazdu  
 )  
 BEGIN  
 RAISERROR ('Pojazd o podanym ID nie istnieje', 16, 4)  
 RETURN;  
 END  
  
 IF NOT EXISTS (  
 SELECT K.ID\_kierowcy  
 FROM Kierowcy AS K  
 WHERE K.ID\_kierowcy = @ID\_kierowcy  
 )  
 BEGIN  
 RAISERROR ('Kierowca o podanym ID nie istnieje', 16, 5)  
 RETURN;  
 END  
  
 IF EXISTS (  
 SELECT R.Numer\_przystanku  
 FROM @Rozkład AS R  
 LEFT JOIN Przystanki AS P  
 ON P.Nazwa\_przystanku = R.Nazwa\_przystanku  
 WHERE P.Nazwa\_przystanku IS NULL  
 )  
 BEGIN  
 RAISERROR ('Niektóre z podanych przystanków nie istnieją', 16, 6)  
 RETURN;  
 END  
  
 BEGIN  
 DECLARE @Pierwszy\_przystanek NVARCHAR(50)  
 DECLARE @Ostatni\_przystanek NVARCHAR(50)  
 SET @Pierwszy\_przystanek = (  
 SELECT TOP 1 Nazwa\_przystanku  
 FROM @Rozkład  
 ORDER BY Numer\_przystanku   
 )  
 SET @Ostatni\_przystanek = (  
 SELECT TOP 1 Nazwa\_przystanku  
 FROM @Rozkład  
 ORDER BY Numer\_przystanku DESC  
 )  
  
 INSERT INTO Kursy  
 VALUES (@Nr\_kursu, @Nr\_linii, @ID\_pojazdu, @Pierwszy\_przystanek, @Ostatni\_przystanek, @ID\_kierowcy)  
  
 INSERT INTO Rozkłady\_jazdy  
 SELECT @Nr\_linii, R.Nazwa\_przystanku, @Nr\_kursu, R.Godzina\_odjazdu  
 FROM @Rozkład AS R  
 END

[link do kodu](../../procs/Dodaj_kurs.sql)

# Dodaj\_linię

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| Numer\_lini | INT |
| Typ\_pojazdu | NVARCHAR(50) |

## Opis działania

Sprawdź czy numer linii jest w poprawnym zakresie:

* [1, 99] dla tramwajów,
* [100, 999] dla autobusów.

i dodaj linię do bazy danych.

## Kod źródłowy

CREATE OR ALTER PROCEDURE Dodaj\_linię   
 @Numer\_lini INT,   
 @Typ\_pojazdu NVARCHAR(50)  
AS  
 SET NOCOUNT ON  
 IF EXISTS (  
 SELECT L.Numer\_linii  
 FROM Linie AS L  
 WHERE L.Numer\_linii = @Numer\_lini  
 )  
 BEGIN  
 RAISERROR ('Linia o takim numerze już istnieje', 16, 1)  
 RETURN;  
 END  
  
 IF (@Typ\_pojazdu = 'autobus' AND (@Numer\_lini BETWEEN 100 AND 999)) OR (@Typ\_pojazdu = 'tramwaj' AND (@Numer\_lini BETWEEN 1 AND 99))  
 BEGIN  
 INSERT INTO Linie  
 VALUES (@Numer\_lini, @Typ\_pojazdu)  
 END  
 ELSE  
 RAISERROR ('Niepoprawny numer linii', 16, 2)  
 RETURN;

[link do kodu](../../procs/Dodaj_linię.sql)

# Dodaj\_mandat

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| Id\_mandatu | INT |
| Imię | NVARCHAR(50) |
| Nazwisko | NVARCHAR(50) |
| PESEL | BIGINT |
| Adres | NVARCHAR(150) |
| Kwota | MONEY |
| Id\_kontrolera | INT |
| Data\_wystawienia | DATE |

## Opis działania

Jeśli dodawany jest mandat o istniejącym już Id\_mandatu to jest błąd. Jeśli próbujemy wstawić mandat na nieistniejącego kontrolera to jest błąd. W przeciwnym wypadku wstawiamy mandat o podanych parametrach.

## Kod źródłowy

CREATE OR ALTER PROCEDURE Dodaj\_mandat   
 @Id\_mandatu INT,  
 @Imię nvarchar(50),  
 @Nazwisko nvarchar(50),  
 @PESEL BIGINT,  
 @Adres nvarchar(150),  
 @Kwota MONEY,  
 @Id\_kontrolera INT,  
 @Data\_wystawienia DATE  
AS  
 IF EXISTS (  
 SELECT \*  
 FROM Mandaty  
 WHERE ID\_mandatu = @Id\_mandatu)  
 BEGIN  
 RAISERROR('Mandat o takim ID już istnieje',16,1)  
 RETURN;  
 END  
 ELSE IF NOT EXISTS (  
 SELECT \*   
 FROM Kontrolerzy  
 WHERE ID\_kontrolera = @Id\_kontrolera)  
 AND @Id\_kontrolera IS NOT NULL  
 BEGIN  
 RAISERROR('Próbujesz wystawić mandat na nieistniejącego kontrolera',16,2)  
 RETURN;  
 END  
 ELSE  
 BEGIN  
 INSERT INTO Mandaty  
 VALUES(@Id\_mandatu,@Imię,@Nazwisko,@PESEL,@Adres,@Kwota,@Id\_kontrolera,NULL,@Data\_wystawienia)  
 END

[link do kodu](../../procs/Dodaj_mandat.sql)

# Dodaj\_pojazd

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| ID\_pojazdu | INT |
| Typ\_pojazdu | NVARCHAR(50) |
| Producent | NVARCHAR(50) |
| Model | NVARCHAR(50) |
| Data\_zakupu | DATE |
| Numer\_rejestracyjny | NCHAR(10) |
| ID\_zajezdni | INT |

## Opis działania

Dodaj do bazy danych nowy pojazd o podanych parametrach. Sprawdź czy w zajezdni, do której pojazd jest dodawany, jest wolne miejsce, oraz zaktualizuj pojemnośc zajezdni. Jeżeli nie podano ID\_zajezdni to dodaj pojazd do magazynu (zajezdni o ID\_zajezdni = 1).

## Kod źródłowy

CREATE OR ALTER PROCEDURE Dodaj\_pojazd  
 @ID\_pojazdu INT,  
 @Typ\_pojazdu NVARCHAR(50),  
 @Producent NVARCHAR(50),  
 @Model NVARCHAR(50),  
 @Data\_zakupu DATE,  
 @Numer\_rejestracyjny NCHAR(10) = NULL,  
 @ID\_zajezdni INT = NULL  
AS  
 SET NOCOUNT ON  
 IF @ID\_zajezdni IS NOT NULL AND EXISTS (  
 SELECT \*  
 FROM Zajezdnie AS Z  
 WHERE Z.ID\_zajezdni = @ID\_zajezdni  
 )  
 BEGIN  
 DECLARE @miejsce int = 0;  
 IF @Typ\_pojazdu = 'autobus'  
 SELECT @miejsce = (  
 SELECT (Pojemność\_autobusów - Liczba\_autobusów) AS Miejsce  
 FROM Zajezdnie AS Z  
 WHERE Z.ID\_zajezdni = @ID\_zajezdni  
 )  
 ELSE  
 SELECT @miejsce = (  
 SELECT (Pojemność\_tramwajów - Liczba\_tramwajów) AS Miejsce  
 FROM Zajezdnie AS Z  
 WHERE Z.ID\_zajezdni = @ID\_zajezdni  
 )  
 IF @miejsce > 0  
 BEGIN  
 INSERT INTO Pojazdy  
 VALUES  
 (@ID\_pojazdu, @Typ\_pojazdu, @Producent, @Model, @Data\_zakupu, @Numer\_rejestracyjny, @ID\_zajezdni);  
  
 IF @Typ\_pojazdu = 'autobus'  
 BEGIN  
 UPDATE Zajezdnie  
 SET Liczba\_autobusów = Liczba\_autobusów + 1  
 WHERE ID\_zajezdni = @ID\_zajezdni  
 END  
 ELSE  
 BEGIN  
 UPDATE Zajezdnie  
 SET Liczba\_tramwajów = Liczba\_tramwajów + 1  
 WHERE ID\_zajezdni = @ID\_zajezdni  
 END  
 END  
 ELSE  
 RAISERROR ('Wybrana zajezdnia jest pełna', 1, 1)  
 RETURN  
 END  
 ELSE  
 BEGIN  
 INSERT INTO Pojazdy  
 VALUES  
 (@ID\_pojazdu, @Typ\_pojazdu, @Producent, @Model, @Data\_zakupu, @Numer\_rejestracyjny, NULL);  
 END

[link do kodu](../../procs/Dodaj_pojazd.sql)

# Przenieś\_pojazd

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| ID\_pojazdu | INT |
| ID\_zajezdni\_początkowej | INT |
| ID\_zajezdni\_docelowej | INT |

## Opis działania

Przenieś pojazd między zajezdniami i zaktualizuj stan zapełnienia odpowiednich zajezdni.

## Kod źródłowy

CREATE OR ALTER PROCEDURE Przenieś\_pojazd  
 @ID\_pojazdu INT,  
 @ID\_zajezdni\_początkowej INT,  
 @ID\_zajezdni\_docelowej INT  
AS  
 IF NOT EXISTS (  
 SELECT P.ID\_pojazdu  
 FROM Pojazdy AS P  
 WHERE (P.ID\_pojazdu = @ID\_pojazdu) AND (P.ID\_zajezdni = @ID\_zajezdni\_początkowej)  
 )  
 BEGIN  
 RAISERROR('Podanego pojazdu nie ma w zajezdni', 16, 1)  
 RETURN  
 END  
  
 IF NOT EXISTS (  
 SELECT Z.ID\_zajezdni  
 FROM Zajezdnie AS Z  
 WHERE @ID\_zajezdni\_docelowej = Z.ID\_zajezdni  
 )  
 BEGIN  
 RAISERROR ('Zajezdnia początkowa nie istnieje', 16, 2)  
 RETURN  
 END  
  
 IF NOT EXISTS (  
 SELECT Z.ID\_zajezdni  
 FROM Zajezdnie AS Z  
 WHERE @ID\_zajezdni\_docelowej = Z.ID\_zajezdni  
 )  
 BEGIN  
 RAISERROR ('Zajezdnia docelowa nie istnieje', 16, 3)  
 RETURN  
 END  
  
 IF NOT EXISTS (  
 SELECT P.ID\_pojazdu  
 FROM Pojazdy AS P  
 WHERE @ID\_pojazdu = P.ID\_pojazdu  
 )  
 BEGIN  
 RAISERROR ('Pojazd nie istnieje', 16, 4)  
 RETURN  
 END  
  
 IF @ID\_zajezdni\_docelowej = @ID\_zajezdni\_początkowej  
 BEGIN  
 RAISERROR('Zajednia początkowa nie może być taka sama jak docelowa', 16, 5)  
 RETURN  
 END  
  
 DECLARE @Typ\_pojazdu VARCHAR(50)  
 SET @Typ\_pojazdu = (  
 SELECT Typ\_pojazdu  
 FROM Pojazdy AS P  
 WHERE @ID\_pojazdu = P.ID\_pojazdu  
 )  
  
 IF @Typ\_pojazdu = 'autobus'  
 BEGIN  
 DECLARE @Pojemność\_autobusów INT  
 SET @Pojemność\_autobusów = (  
 SELECT (Z.Pojemność\_autobusów - Z.Liczba\_autobusów)  
 FROM Zajezdnie AS Z  
 WHERE Z.ID\_zajezdni = @ID\_zajezdni\_docelowej  
 )  
  
 IF @Pojemność\_autobusów <= 0  
 BEGIN  
 RAISERROR ('Zajezdnia docelowa jest pełna', 16, 6)  
 RETURN  
 END  
  
 UPDATE Zajezdnie  
 SET Liczba\_autobusów =   
 (  
 CASE  
 WHEN ID\_zajezdni = @ID\_zajezdni\_początkowej THEN (Liczba\_autobusów - 1)  
 WHEN ID\_zajezdni = @ID\_zajezdni\_docelowej THEN (Liczba\_autobusów + 1)  
 ELSE Liczba\_autobusów  
 END  
 )  
 UPDATE Pojazdy  
 SET ID\_zajezdni = @ID\_zajezdni\_docelowej  
 WHERE ID\_pojazdu = @ID\_pojazdu  
 END  
 ELSE  
 BEGIN  
 DECLARE @Pojemność\_tramwajów INT  
 SET @Pojemność\_tramwajów = (  
 SELECT (Z.Pojemność\_tramwajów - Z.Pojemność\_tramwajów)  
 FROM Zajezdnie AS Z  
 WHERE Z.ID\_zajezdni = @ID\_zajezdni\_docelowej  
 )  
  
 IF @Pojemność\_tramwajów <= 0  
 BEGIN  
 RAISERROR ('Zajezdnia docelowa jest pełna', 16, 6)  
 RETURN  
 END  
   
 UPDATE Zajezdnie  
 SET Liczba\_tramwajów =   
 (  
 CASE  
 WHEN ID\_zajezdni = @ID\_zajezdni\_początkowej THEN (Liczba\_tramwajów - 1)  
 WHEN ID\_zajezdni = @ID\_zajezdni\_docelowej THEN (Liczba\_tramwajów + 1)  
 ELSE Liczba\_autobusów  
 END  
 )  
 UPDATE Pojazdy  
 SET ID\_zajezdni = @ID\_zajezdni\_docelowej  
 WHERE ID\_pojazdu = @ID\_pojazdu  
 END

[link do kodu](../../procs/Przenieś_pojazd.sql)

# Usuń\_pojazd

## Parametry wejściowe

|  |  |
| --- | --- |
| Nazwa | Typ |
| ID\_pojazdu | INT |

## Opis działania

Usuń z tabeli Pojazd pojazd o podanym ID\_pojazdu oraz zaktualizuj pojemności odpowiednich zajezdni.

## Kod źródłowy

CREATE OR ALTER PROCEDURE Usuń\_pojazd @ID\_pojazdu INT  
AS  
 SET NOCOUNT ON  
 IF NOT EXISTS (SELECT \*  
 FROM Pojazdy  
 WHERE ID\_pojazdu = @ID\_pojazdu  
 )  
 BEGIN  
 RAISERROR('Pojazd nie istnieje', 16, 1)  
 RETURN  
 END  
  
 DECLARE @Typ\_pojazdu VARCHAR(50)  
 SELECT @Typ\_pojazdu = (  
 SELECT Typ\_pojazdu  
 FROM Pojazdy AS P  
 WHERE P.ID\_pojazdu = @ID\_pojazdu  
 )  
  
 DECLARE @ID\_zajezdni INT  
 SET @ID\_zajezdni = (  
 SELECT ID\_zajezdni  
 FROM Pojazdy AS P  
 WHERE P.ID\_pojazdu = @ID\_pojazdu  
 )  
  
 IF @Typ\_pojazdu = 'autobus'  
 BEGIN  
 UPDATE Zajezdnie  
 SET Liczba\_autobusów = Liczba\_autobusów - 1  
 WHERE ID\_zajezdni = @ID\_zajezdni  
 END  
 ELSE  
 BEGIN  
 UPDATE Zajezdnie  
 SET Liczba\_tramwajów = Liczba\_tramwajów - 1  
 WHERE ID\_zajezdni = @ID\_zajezdni  
 END  
  
 BEGIN  
 DELETE FROM Pojazdy  
 WHERE ID\_pojazdu = @ID\_pojazdu  
 END

[link do kodu](../../procs/Usuń_pojazd.sql)

# Bilety

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **ID\_biletu (PK)** | INT | Nie | - | PRIMARY KEY, IDENTITY(1,1) | - | ID biletu |
| ID\_klienta | INT | Nie | - | - | Klienci.ID\_klienta | ID klienta, który kupił bilet |
| Rodzaj\_biletu | NVARCHAR(50) | Nie | - | - | Rodzaje\_biletów.Rodzaj\_biletu | Rodzaj biletu (normalny, ulgowy, itd.) |
| Cena | MONEY | Nie | - | - | - | Rzeczywista cena biletu |
| Data\_zakupu | DATE | Nie | - | - | - | Data zakupu biletu |
| Data\_wygaśnięcia | DATE | Nie | - | - | - | Data utraty ważności biletu |
| ID\_punktu\_sprzedaży | INT | Tak | - | - | Punkty\_sprzedaży.ID\_punktu | Data utraty ważności biletu |

## Wyzwalacze

Blokada\_zakupu\_biletu AFTER INSERT

## Opis tabeli

Tabela zawiera informacje o aktywnych i wygasłych biletach oraz o ich rzeczywistych cenach (uwzględniających wszystkie ulgi). Cena biletów zmienia się w czasie – aktualna cena biletów jest przechowywana w tabeli Rodzaje\_biletów, a przy zakupie biletu jest mu przypisywana jego aktualna cena. Po zmianie ceny w Rodzaje\_biletów, cena w Bilety pozostaje bez zmian.

# Czas\_pracy

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **ID\_kierowcy (PK)** | INT | Nie | - | PRIMARY KEY | Kierowcy.ID\_kierowcy | ID kierowcy |
| Rok | INT | Nie | - | - |  | Rok wpisu |
| Miesiąc | INT | Nie | - | CHECK BETWEEN 1 AND 12 |  | Miesiąc wpisu |
| Godziny\_pracy | INT | Nie | - | - |  | Liczba przepracowanych w miesiącu godzin |
| Nadgodziny | INT | Nie | - | - |  | Liczba przepracowanych w miesiącu nadgodzin |

## Opis tabeli

Tabela zawiera historię przepracowanych godzin przez wszystkich pracowników z podziałem na lata i miesiące.

# Kierowcy

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **ID\_Kierowcy (PK)** | INT | Nie | - | PRIMARY KEY, IDENTITY(1,1) | - | ID kierowcy |
| Imię | NVARCHAR(50) | Nie | - | - | - | Imię kierowcy |
| Nazwisko | NVARCHAR(50) | Nie | - | - | - | Nazwisko kierowcy |
| Data\_urodzenia | DATE | Nie | - | - | - | Data urodzenia |
| PESEL | BIGINT | Nie | - | CHECK PESEL >= 10000000000 AND PESEL<= 99999999999, UNIQUE | - | Data urodzenia |

## Opis tabeli

Zawiera podstawowe informacje o kierowcach.

# Klienci

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **ID\_klienta (PK)** | INT | Nie | - | PRIMARY KEY, IDENTITY | - | ID osoby |
| Imię | NVARCHAR(50) | Nie | - | - | - | Imię klienta |
| Nazwisko | NVARCHAR(50) | Nie | - | - | - | Nazwisko klienta |
| PESEL | BIGINT | Nie | - | CHECK PESEL >= 10000000000 AND PESEL<= 99999999999 | - | PESEL klienta |
| Ulga | NVARCHAR(50) | Tak | - | - | Ulgi.Rodzaj\_ulgi | Rodzaj ulgii przysługującej klientowi |

## Opis tabeli

Tabela zawiera podstawowe informacje o klientach.

# Kontrolerzy

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **ID\_kontrolera (PK)** | INT | Nie | - | PRIMARY KEY, IDENTITY(1,1) | - | ID kontrolera |
| Imię | NVARCHAR(50) | Nie | - | - | - | Imię kontrolera |
| Nazwisko | NVARCHAR(50) | Nie | - | - | - | Nazwisko kontrolera |

## Opis tabeli

Tabela zawiera informacje o kontrolerach biletów.

# Kursy

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **Nr\_kursu (PK)** | INT | Nie | - | PRIMARY KEY | - | Numer kursu |
| **Numer\_linii (PK)** | INT | Nie | - | PRIMARY KEY | Linie.Numer\_linii | Numer linii, której dotyczy kurs |
| ID\_pojazdu | INT | Tak | - | - | Pojazdy.ID\_pojazdu | ID\_pojazdu przypisanego do kursu |
| Pierwszy\_przystanek | NVARCHAR(50) | Nie | - | - | Przystanki.Nazwa\_przystanku | Nazwa pierwszego przystanku w kursie |
| Ostatni\_przystanek | NVARCHAR(50) | Nie | - | - | Przystanki.Nazwa\_przystanku | Nazwa ostatniego przystanku w kursie |
| ID\_kierowcy | INT | Nie | - | - | Kierowcy.ID\_kierowcy | ID\_kierowcy, który będzie prowadził pojazd w danym kursie |

## Wyzwalacze

Blokada\_dodania\_niezarejstrowanych pojazdów AFTER INSERT

## Opis tabeli

Tabela zawiera podstawowe informacje o pojedynczym kursie linii.

# Linie

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **Numer\_linii (PK)** | INT | Nie | - | PRIMARY KEY, CHECK ((Typ\_pojazdu LIKE 'tramwaj' AND Numer\_linii BETWEEN 0 AND 100) OR (Typ\_pojazdu LIKE 'autobus' AND Numer\_linii BETWEEN 100 AND 999)) | - | Numer linii |
| Typ\_pojazdu | NVARCHAR(50) | Nie | - | - | - | Typ linii |

## Opis tabeli

Tabela zawiera podstawowe informacje o liniach.

# Mandaty

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **ID\_mandatu (PK)** | INT | Nie | - | PRIMARY KEY | - | ID mandatu |
| Imię | NVARCHAR(50) | Nie | - | - | - | Imię ukaranego |
| Imię | NVARCHAR(50) | Nie | - | - | - | Nazwisko ukaranego |
| PESEL | BIGINT | Nie | - | CHECK PESEL >= 10000000000 AND PESEL<= 99999999999 | - | PESEL ukaranego |
| Adres | NVARCHAR(150) | Nie | - | - | - | Adres ukaranego |
| Kwota | MONEY | Nie | - | - | - | Kwota mandatu |
| ID\_wystawiającego | INT | Tak | - | - | Kontrolerzy.ID\_kontrolera | ID kontrolera wystawiającego mandat |
| ID\_klienta | INT | Tak | - | - | Klienci.ID\_klienta | ID klienta, który otrzymał mandat |
| Data\_wystawienia | DATE | Nie | - | - | - | Data wystawienia mandatu |

## Opis tabeli

Tabela zawiera informacje o wystawionych mandatach oraz opcjonalnie o klientach które je otrzymali.

# Pojazdy

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **ID\_pojazdu (PK)** | INT | Nie | - | PRIMARY KEY | - | ID pojazdu |
| Typ\_pojazdu | NVARCHAR(50) | Nie | - | CHECK 'autobus' OR 'tramwaj' | - | Typ pojazdu |
| Producent | NVARCHAR(50) | Nie | - | - | - | Producent pojazdu |
| Model | NVARCHAR(50) | Nie | - | - | - | Model pojazdu |
| Data\_zakupu | DATE | Nie | - | - | - | Model pojazdu |
| Numer\_rejestracyjny | NCHAR(10) | Tak | - | - | - | Numer rejestracyjny pojazdu |
| ID\_zajezdni | INT | Tak | - | - | Zajezdnie.ID\_zajezdni | ID zajezdni, do której aktualnie przypisany jest pojazd |

## Wyzwalacze

Dodaj\_przegląd AFTER INSERT

## Opis tabeli

Tabela zawiera podstawowe informacje o pojazdach.

# Nazwa tabeli

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **ID\_pojazdu (PK)** | INT | Nie | - | PRIMARY KEY | Pojazdy.ID\_pojazdu | ID pojazdu |
| Data\_przeglądu | DATE | Nie | - | CHECK Data\_wygaśnięcia >= Data\_przeglądu | - | Data przeglądu |
| Data\_wygaśnięcia | DATE | Nie | - | CHECK Data\_wygaśnięcia >= Data\_przeglądu | - | Data wygaśnięcia przeglądu |

## Opis tabeli

Tabela zawiera informacje o ważności przeglądów wszystkich pojazdów.

# Przystanki

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **Nazwa\_przystanku (PK)** | NVARCHAR(50) | Nie | - | PRIMARY KEY | - | ID przystanku |
| Współrzędne\_geograficzne | NVARCHAR(50) | Nie | - | - | - | Dokładna lokalizacja przystanku |
| Kod\_typu\_przystanku | NCHAR(3) | Nie | - | - | Typy\_przystanków.Kod\_typu | Kod identyfikujący typ przystanku (tramwajowy, autobusowy, itp.) |
| Czy\_pętla | BIT | Tak | - | - | - | Czy przystanek jest pętlą |

## Opis tabeli

Tabela zawiera informacje o przystankach i ich lokalizacjach.

# Punkty\_sprzedaży

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **ID\_punktu (PK)** | INT | Nie | - | PRIMARY KEY, IDENTITY(1,1) | - | ID punktu sprzedaży |
| Nazwa\_przystanku | NVARCHAR(50) | Nie | - | - | Przystanki.Nazwa\_przystanku | Nazwa przystanku, na którym znajduje się punkt sprzedaży |

## Opis tabeli

Tabela zawiera informacje o punktach sprzedaży oraz ich lokalizacjach.

# Rodzaje\_biletów

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **Rodzaj\_biletu (PK)** | INT | Nie | - | PRIMARY KEY, IDENTITY | tab2.ID | Typ biletu (normalny, ulgowy, itd.) |
| Cena\_bazowa | MONEY | Nie | - | - | - | Podstawowa cena biletu |
| Okres\_w\_dniach | INT | Nie | - | - | - | Okres ważności biletu w dniach |

## Opis tabeli

Tabela zawiera podstawowe informacje o dostępnych rodzajach biletów (normalny, ulgowy, itd.).

# Nazwa tabeli

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **Numer\_linii (PK)** | INT | Nie | - | PRIMARY KEY | Linie.Numer\_linii | Numer linii, której dotyczy wpis |
| **Nazwa\_przystanku (PK)** | NVARCHAR(50) | Nie | - | PRIMARY KEY | Przystanki.Nazwa\_przystanku | Nazwa przystanku |
| **Numer\_kursu (PK)** | INT | Nie | - | PRIMARY KEY | Kursy.Numer\_kursu | Nazwa kursu, którego dotyczy wpis |
| Godzina\_odjazdu | TIME(0) | Nie | - | - | - | Godzina odjazdu z przystanku |

## Opis tabeli

Tabela zawiera rozkłady jazdy wszystkich kursów wszystkich linii w bazie danych.

# Typy\_przystanków

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| Typ | NVARCHAR(50) | Nie | - | - | - | Typ przystanku (tramwajowy, autobusowy) |
| **Kod\_typu (PK)** | NCHAR(3) | Nie | - | - | - | Kod typu przystanku |

## Opis tabeli

Tabela zawiera możliwe typy przystanków.

# Ulgi

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **Rodzaj\_ulgi (PK)** | NVARCHAR(50) | Nie | - | PRIMARY KEY |  | Rodzaj ulgi |
| Zniżka | INT | Tak | - | CHECK BETWEEN 0 AND 100 | - | Wartośc ulgii w procentach |

## Opis tabeli

Tabela zawiera informacje o możliwych ulgach i przysługjących w związku z nimi ulgach.

# Zajezdnie

## Kolumny

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nazwa | Typ | Nullable | Default | Constraints | Klucze obce | Opis |
| **ID\_zajezdni (PK)** | INT | Nie | - | PRIMARY KEY | - | ID zajezdni |
| Nazwa | NVARCHAR(50) | Nie | - | - | - | Nazwa zajezdni |
| Adres | NVARCHAR(50) | Nie | - | - | - | Adres zajezdni |
| Pojemność\_autobusów | INT | Tak | - | - | - | Maksymalna pojemność autobusów |
| Pojemność\_tramwajów | INT | Tak | - | - | - | Maksymalna pojemność tramwajów |
| Liczba\_autobusów | INT | Tak | - | - | - | Liczba autobusów obecnie przypisanych do zajezdni |
| Liczba\_tramwajów | INT | Tak | - | - | - | Liczba pojazdów obecnie przypisanych do zajezdni |

## Opis tabeli

Tabela zawiera informacje o zajezdniach i ich aktualnym stanie zapełnienia.

# Blokada\_dodania\_niezarejestrowanych\_pojazdów

## Warunek uruchomienia

ON Kursy AFTER INSERT

## Opis działania

Zablokuj możliwość dodawania do kursów pojazdów, które nie mają ustawionego numeru rejestracyjnego.

## Kod źródłowy

CREATE OR ALTER TRIGGER Blokada\_dodania\_niezarejestrowanych\_pojazdów ON Kursy  
AFTER INSERT  
AS  
IF (ROWCOUNT\_BIG() = 0)  
RETURN;  
IF EXISTS (  
 SELECT \*  
 FROM inserted AS I  
 INNER JOIN Pojazdy AS P  
 ON P.ID\_pojazdu = I.ID\_pojazdu  
 WHERE P.Numer\_rejestracyjny IS NULL  
)  
BEGIN  
 RAISERROR ('Nie można dodać niezarejestrowanego pojazdu do linii', 16, 1)  
 ROLLBACK TRANSACTION  
 RETURN  
END;  
GO

[link do kodu](../../triggers/Blokada_dodania_niezarejestrowanych_pojazdów.sql)

# Blokada\_zakupu\_biletu

## Warunek uruchomienia

ON Bilety AFTER INSERT

## Opis działania

Zablokuj możliwość dodania biletu dla osoby z niezapłaconymi mandatami.

## Kod źródłowy

CREATE OR ALTER TRIGGER Blokada\_zakupu\_biletu ON Bilety  
AFTER INSERT  
AS  
IF (ROWCOUNT\_BIG() = 0)  
RETURN;  
IF EXISTS(  
 SELECT \*  
 FROM Mandaty AS M  
 INNER JOIN inserted AS I  
 ON I.ID\_klienta = M.ID\_klienta  
)  
BEGIN  
 RAISERROR ('Dodano bilet dla osoby z niezapłaconymi mandatami', 1, 1);  
 ROLLBACK TRANSACTION;  
 RETURN  
END;  
GO

[link do kodu](../../triggers/Blokada_zakupu_biletu.sql)

# Dodaj\_przegląd

## Warunek uruchomienia

ON Pojazdy AFTER INSERT

## Opis działania

Po dodaniu wierszy nowych pojazdów do tabeli Pojazdy, wstaw do tabeli Przeglądy planowaną datę przeglądu pojazdu.

## Kod źródłowy

CREATE OR ALTER TRIGGER Dodaj\_przegląd ON Pojazdy  
AFTER INSERT  
AS  
IF (ROWCOUNT\_BIG() = 0)  
RETURN;  
INSERT INTO Przeglądy  
SELECT I.ID\_pojazdu, I.Data\_zakupu, (DATEADD(DAY, 180, I.Data\_zakupu))  
FROM inserted I  
GO

[link do kodu](../../triggers/Dodaj_przegląd.sql)

# wstawienie\_Id\_klienta\_przez\_pesel

## Warunek uruchomienia

ON Mandaty AFTER INSERT

## Opis działania

Wystawiający mandat nie zna ID klienta, stąd też wyzwalacz który zaktualizuje Id\_klienta w tabeli Mandaty na podstawie wpisanego Peselu.

## Kod źródłowy

CREATE OR ALTER TRIGGER wstawienie\_Id\_klienta\_przez\_pesel ON Mandaty  
AFTER INSERT  
AS  
 IF (ROWCOUNT\_BIG() = 0)  
 RETURN;  
 UPDATE Mandaty  
 SET Mandaty.Id\_klienta = (  
 CASE WHEN EXISTS (  
 SELECT ID\_klienta   
 FROM Klienci   
 WHERE I.PESEL = Klienci.PESEL)   
 THEN (  
 SELECT ID\_klienta   
 FROM Klienci   
 WHERE I.PESEL = Klienci.PESEL)   
 ELSE NULL END)  
 FROM Mandaty AS M  
 INNER JOIN inserted AS I  
 ON I.ID\_mandatu = M.ID\_mandatu

[link do kodu](../../triggers/wstawienie_Id_klienta_przez_pesel.sql)

# wyliczanie\_ceny

## Warunek uruchomienia

ON Bilety AFTER INSERT

## Opis działania

Zaktualizuj cenę biletu w tabeli Bilety o przynależną osobie kupującej bilet ulgę.

## Kod źródłowy

CREATE OR ALTER TRIGGER wyliczanie\_ceny ON Bilety  
AFTER INSERT  
AS  
 IF (ROWCOUNT\_BIG() = 0)  
 RETURN;  
 UPDATE Bilety  
 SET cena = (CASE WHEN Ulga IS NOT NULL THEN (ROUND(Cena\_bazowa\*CAST((1-U.Zniżka/100.0) as MONEY),2)) ELSE R.Cena\_bazowa END)  
 FROM Bilety AS B  
 INNER JOIN Rodzaje\_biletów AS R  
 ON R.Rodzaj\_biletu = B.Rodzaj\_biletu  
 INNER JOIN Klienci AS K  
 ON B.ID\_klienta = K.ID\_klienta  
 LEFT JOIN Ulgi AS U  
 ON U.Rodzaj\_ulgi = K.Ulga  
 INNER JOIN inserted AS I  
 ON I.ID\_biletu = B.ID\_biletu

[link do kodu](../../triggers/wyliczanie_ceny.sql)

# wyliczanie\_daty\_wygasniecia\_biletu

## Warunek uruchomienia

ON Bilety AFTER INSERT

## Opis działania

Zaktualizuj dodany bilet w tabeli Bilety o datę wygaśnięcia biletu wyliczaną na bazie daty zakupu + liczbie dni ważności biletu na podstawie kupowanego rodzaju biletu z tabeli Rodzaje\_biletów.

## Kod źródłowy

CREATE OR ALTER TRIGGER wyliczanie\_daty\_wygasniecia\_biletu ON Bilety  
AFTER INSERT  
AS  
 IF (ROWCOUNT\_BIG() = 0)  
 RETURN;  
 UPDATE Bilety  
 SET Data\_wygaśnięcia = DATEADD (DD, R.Okres\_w\_dniach, I.Data\_zakupu)  
 FROM Bilety AS B  
 JOIN Rodzaje\_biletów AS R   
 ON B.Rodzaj\_biletu = R.Rodzaj\_biletu  
 JOIN inserted AS I   
 ON I.ID\_biletu = B.ID\_biletu

[link do kodu](../../triggers/wyliczanie_daty_wygasniecia_biletu.sql)

# Pojazdy\_od\_producenta

## Opis działania

Wyświetl wszystkich producentów posiadanych pojazdów, razem z podliczeniem całkowitej liczby pojazdów.

## Kod źródłowy

CREATE OR ALTER VIEW Pojazdy\_od\_producenta  
AS  
SELECT P.Producent, COUNT(\*) AS Ilość  
FROM Pojazdy AS P  
GROUP BY P.Producent

[link do kodu](../../views/Pojazdy_od_producenta.sql)

# Ranking\_Kierowców

## Opis działania

Wypisz ranking wszystkich pracowników, ich dane oraz czas pracy w każdym miesiącu według łącznej ilości przepracowanych godzin i nadgodzin.

## Kod źródłowy

CREATE VIEW Ranking\_Kierowców as  
SELECT Rok, Miesiąc, ROW\_NUMBER()   
 OVER(PARTITION BY Rok,Miesiąc ORDER BY Godziny\_pracy+Nadgodziny DESC) as Ranking\_Miesiąca,  
K.ID\_kierowcy, Imię, Nazwisko, Data\_urodzenia, PESEL, Godziny\_pracy,Nadgodziny,Godziny\_pracy+Nadgodziny AS Sumaryczna\_Praca   
FROM Kierowcy AS K  
JOIN Czas\_pracy AS C   
 ON K.ID\_kierowcy = C.ID\_kierowcy

[link do kodu](../../views/Ranking_Kierowców.sql)

# Sprzedaż\_biletów

## Opis działania

Wybierz z tabeli Bilety sumę zysku ze sprzedanych biletów, pogrupowane według lat i miesięcy.

## Kod źródłowy

CREATE OR ALTER VIEW Sprzedaż\_biletów  
AS  
SELECT YEAR(B.Data\_zakupu) AS Rok, MONTH(B.Data\_zakupu) AS Miesiąc, SUM(B.Cena) AS Suma\_sprzedanych\_biletów   
FROM Bilety AS B  
INNER JOIN Punkty\_sprzedaży AS P  
 ON P.ID\_punktu = B.ID\_punktu\_sprzedaży  
GROUP BY YEAR(B.Data\_zakupu), MONTH(B.Data\_zakupu)

[link do kodu](../../views/Sprzedaż_biletów.sql)

# Statystyki\_mandatów\_kontrolerów

## Opis działania

Tabela z danymi kontrolera, wystawioną liczbą mandatów oraz łącznym zyskiem z mandatów uzyskanym w ciągu każdego dnia pracy kontrolera.

## Kod źródłowy

CREATE OR ALTER VIEW Statystyki\_mandatów\_kontrolerów AS  
SELECT K.ID\_kontrolera,K.Imie,K.Nazwisko, COUNT(Kwota) AS [Liczba danych mandatów], SUM(Kwota) AS [Suma zł za wszystkie mandaty dane przez kontrolera]   
FROM Kontrolerzy AS K  
JOIN Mandaty AS M   
ON K.ID\_kontrolera = M.ID\_wystawiającego  
GROUP BY K.ID\_kontrolera, K.Imie, K.Nazwisko, M.Data\_wystawienia

[link do kodu](../../views/Statystyki_mandatów_kontrolerów.sql)

# Widok\_zajezdni

## Opis działania

Wypisz wszystkie zajezdnie oraz informacje o nich z tabeli Zajezdnia oraz wszystkie pojazdy znajdujące się w każdej z nich z tabeli Pojazdy.

## Kod źródłowy

CREATE VIEW Widok\_Zajezdni AS  
SELECT Z.ID\_zajezdni, Nazwa, Adres, Pojemność\_autobusów, Pojemność\_tramwajów, Liczba\_autobusów, Liczba\_tramwajów,  
ID\_pojazdu, Typ\_pojazdu, Producent, Model, Data\_zakupu, Numer\_rejestracyjny FROM Zajezdnie as Z  
LEFT JOIN Pojazdy as P on Z.ID\_zajezdni = P.ID\_zajezdni

[link do kodu](../../views/Widok_Zajezdni.sql)

# Wynagrodzenia

## Opis działania

Wyświetl miesięczne wynagrodzenia dla wszystkich pracowników, poprzez pomnożenie czasu pracy i nadgodzin przez z tabeli Czas\_pracy prez stawki obowiązujące w danym miesiącu i roku z tabeli Stawki\_miesięczne.

## Kod źródłowy

CREATE OR ALTER VIEW Wynagrodzenia  
AS  
SELECT C.ID\_kierowcy, C.Rok, C.Miesiąc,  
 CAST((C.Godziny\_pracy \* S.Stawka\_podstawowa + C.Nadgodziny \* S.Stawka\_za\_nadgodziny) AS money) AS Wynagrodzenie  
FROM Czas\_pracy AS C  
LEFT JOIN Stawki\_miesięczne AS S  
 ON (C.Rok = S.Rok) AND (C.Miesiąc = S.Miesiąc)

[link do kodu](../../views/Wynagrodzenia.sql)