



**Kolegium Nauk Przyrodniczych  
Uniwersytet Rzeszowski**

**Przedmiot:  
Bazy Danych 2**

**Projekt Komis Samochodowy**

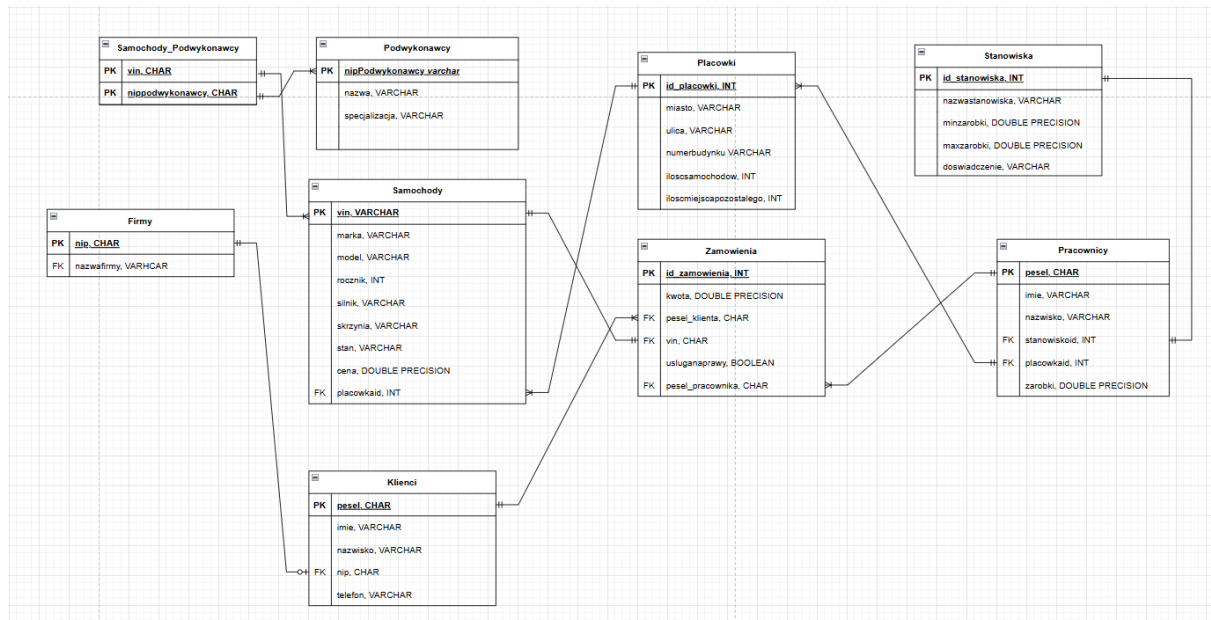
**Wykonał:  
Kacper Dusza, 131427  
Radosław Cebula, 131413**

**Prowadzący: dr. Piotr Grochowalski  
Rzeszów 2025**

# 1. Wstęp

Projekt zakłada stworzenie bazy danych dla przedsiębiorstwa zajmującego się sprzedażą pojazdów oraz zarządzaniem placówkami i podwykonawcami. Celem jest efektywna organizacja danych związanych z klientami, zamówieniami, pracownikami, placówkami oraz pojazdami.

Baza danych została stworzona w oparciu o PostgreSQL. Aby ułatwić dostęp do danych, wdrożono aplikację internetową w technologii PHP, z intuicyjnym interfejsem graficznym. Aplikacja umożliwia przeglądanie, edytowanie, dodawanie i usuwanie danych w tabelach.



## 2. Specyfikacja tematu projektu

Projektowana rzeczywistość obejmuje działalność firmy motoryzacyjnej, która:

- Zarządza placówkami, w których przechowywane są pojazdy i pracownicy.
  - Współpracuje z podwykonawcami odpowiedzialnymi za naprawy i pojazdów.
    - Obsługuje klientów indywidualnych i biznesowych.
  - Realizuje zamówienia na zakup i ewentualną naprawę pojazdu.
    - Zarządza pracownikami oraz ich stanowiskami
    - Przechowuje informacje o klientach i ich zamówieniach.
- 

## 3. Aspekt projektowy bazy danych

Baza danych została zaprojektowana z podziałem na następujące schematy:

- **Klienci:** dane osobowe i kontaktowe klientów.
- **Zamówienia:** szczegóły dotyczące zamówień realizowanych przez firmę.
- **Placówki:** informacje o placówkach firmy, w tym liczba miejsc i liczba pojazdów.
  - **Podwykonawcy:** informacje o firmach współpracujących z przedsiębiorstwem.
- **Pojazdy:** szczegółowe dane o pojazdach w magazynie oraz w serwisie.
  - **Firmy:** informacje o przedsiębiorstwach klientów.
  - **Stanowiska:** szczegóły o stanowiskach pracowników firmy.
  - **Pracownicy:** informacje o pracownikach przedsiębiorstwa.

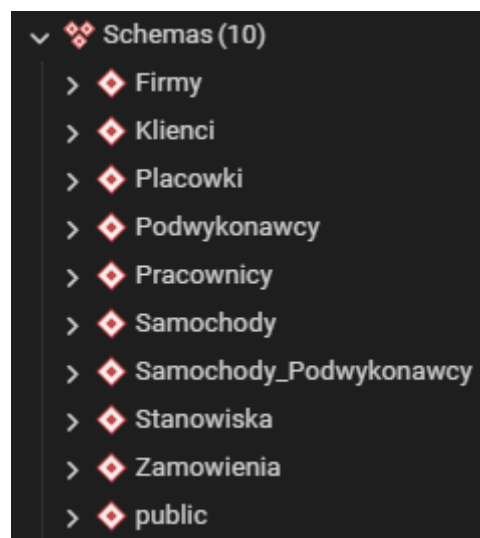
W każdym schemacie zdefiniowano odpowiednie tabele, relacje, funkcje, procedury i triggerzy. Takie podejście zapewnia przejrzystość i optymalizację działania bazy danych.

---

## 4. Aspekt projektowy funkcjonalności bazy danych

### 1. Schematy

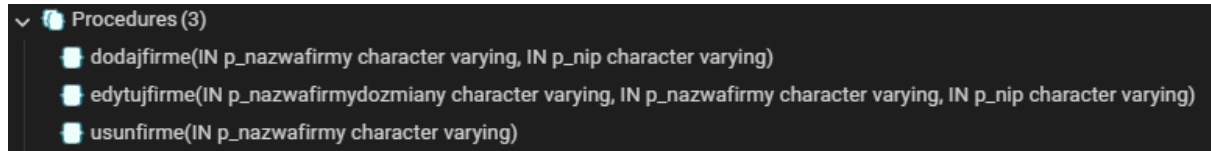
Baza danych została podzielona na schematy. W każdym z nich znajdują się funkcje, triggerzy oraz procedury, które to są związane z daną tabelą. Dzięki takiemu rozwiązaniu baza jest o wiele bardziej przejrzysta i łatwiej się z niej korzysta.



### 2. Procedury

Każda z tabel posiada odrębne procedury które pozwalają na dodawanie, usuwanie i edytowanie danych w tabeli.

*Przykład dla tabeli „firmy”*



### *Dodaj:*

```
BEGIN  
  
INSERT INTO firmy (nazwaFirmy, nip)  
VALUES (p_nazwaFirmy, p_nip);  
  
END;
```

### *Edytuj:*

```
BEGIN  
  
UPDATE firmy  
  
SET nazwaFirmy = p_nazwaFirmy, nip = p_nip  
WHERE nazwaFirmy = p_nazwaFirmyDoZmiany;  
  
END;
```

### *Usuń:*

```
BEGIN  
  
DELETE FROM firmy  
  
WHERE nazwaFirmy = p_nazwaFirmy;  
  
END;
```

### 3. Triggery

Dla tabeli „placówki” zaimplementowano 5 triggerów. Przy ich tworzeniu zostały użyte sekwencje.

- 
- *inkrementacjaLiczbyAut()* – w momencie gdy do danej placówki zostaje dodany samochód, wartość w kolumnie „liczbaSamochodow” zwiększana jest o jeden.

```
BEGIN  
  
UPDATE placowki  
  
SET iloscSamochodow = iloscSamochodow+1  
  
WHERE id_placowki = NEW.placowkaId;  
  
RETURN NEW;  
  
END;
```

---

- *dekrementacjaLiczbyAut()* – w momencie gdy z danej placówki usuwany jest samochód, wartość w kolumnie „liczbaSamochodow” zmniejszana jest o jeden.

```
BEGIN  
  
UPDATE placowki  
  
SET iloscSamochodow = iloscSamochodow -1  
  
WHERE id_placowki = OLD.placowkaId;  
  
RETURN OLD;  
  
END;
```

---

- 
- *inkrementacjaMiejscaPozostalego()* – w momencie gdy z danej placówki usuwany jest samochód, wartość w kolumnie „ilosc\_miejsc\_pozostalego” zostaje zwiększona o jeden.

```
BEGIN

UPDATE placowki

SET ilosc_miejsc_pozostalego =
ilosc_miejsc_pozostalego - 1

WHERE id_placowki = NEW.placowka_id;

RETURN NEW;

END;
```

- 
- *dekrementacjaMiejscaPozostalego()* – w momencie gdy do danej placówki dodawany jest samochód, wartość w kolumnie „ilosc\_miejsc\_pozostalego” jest zmniejszana o jeden.

```
BEGIN

UPDATE placowki

SET ilosc_miejsc_pozostalego =
ilosc_miejsc_pozostalego + 1

WHERE id_placowki = OLD.placowka_id;

RETURN OLD;

END;
```

---

- 
- `check_iloscMiejscaPozostalego()` – w momencie gdy będziemy chcieli dodać samochód do placówki w której wartość kolumny `iloscMiejscaPozostalego` będzie wynosić 0, trigger ten na to nie pozwoli.

```
BEGIN

IF (SELECT ilosc_miejsc_pozostalego FROM placowki
WHERE id_placowki = NEW.placowka_id) = 0 THEN

RAISE EXCEPTION 'Nie można dodać samochodu: brak wolnych miejsc w placówce';

END IF;

RETURN NEW;

END;
```

---





---

## 4. Funkcje

- Funkcje pobierające – każda z tabel posiada oddzielną funkcję, która służy do pobierania danych. Funkcja wykorzystywana jest to pobrania danych na interfejs graficzny.

*Przykład dla tabeli „klienci”*

```
BEGIN
RETURN QUERY
SELECT k.pesel,
k.imie,
k.nazwisko,
k.nip,
k.telefon
FROM klienci k;
END;
```

---

- historia\_podwykonawcy(p\_nip VARCHAR) – funkcja która zwraca zapytanie które wyświetla informacje na temat wszystkich zamówień danego podwykonawcy.

```
BEGIN
RETURN QUERY
SELECT
sp.vin,
sp.nippodwykonawcy
FROM samochody_podwykonawcy as sp
WHERE sp.nippodwykonawcy = p_nip;
END;
```

---

- `oblicz_statystyki_podwykonawcy(nippodwykonawcy_param CHAR)` – funkcja ta zwraca zapytanie które wyświetla sumę wartości wszystkich samochodów nad którymi dany podwykonawca pracował oraz w nowej kolumnie wyświetla uśrednioną wartość samochodu. Takie rozwiązanie może dostarczyć przedsiębiorstwu niezwykle cennych informacji.

```
BEGIN
RETURN QUERY
SELECT
sp.nippodwykonawcy,
SUM(s.cena) AS suma_wartosci_aut,
SUM(s.cena) / COUNT(sp.vin) AS
srednia_wartosc_auta
FROM samochody_podwykonawcy sp
JOIN samochody s ON sp.vin = s.vin
WHERE sp.nippodwykonawcy = nippodwykonawcy_param
GROUP BY sp.nippodwykonawcy;
END;
```

---

- `ilosc_pracownikow(p_placowkaid INT)` – funkcja zwraca zapytanie, które wyświetla ilu pracowników jest przypisanych do danej placówki

```
DECLARE
ilosc NUMERIC := 0;
BEGIN
IF p_placowkaid IS NOT NULL THEN
SELECT COUNT(stanowiskoid) INTO ilosc
FROM pracownicy
WHERE placowkaid = p_placowkaid;
ELSE
SELECT COUNT(stanowiskoid) INTO ilosc
FROM pracownicy;
END IF;
RETURN COALESCE(ilosc, 0);
END;
```

---

- suma\_kwoty\_zamowien\_usluga\_naprawy() – funkcja zwraca wartość wszystkich zamówień, dla których zlecono usługę naprawy pojazdu. Do napisania tej funkcji został użyty kursor, dzięki czemu funkcja będzie działać szybciej.

```
DECLARE
r RECORD;
suma double precision := 0;
cur CURSOR FOR
SELECT Kwota
FROM zamowienia
WHERE usluganaprawy = true;
BEGIN
OPEN cur;

LOOP
FETCH cur INTO r;
EXIT WHEN NOT FOUND;

suma := suma + r.Kwota;
END LOOP;

CLOSE cur;

RETURN suma;
END;
```

---

- `zamowieina_danego_klienta(pesel CHAR)` – funkcja zwraca zapytanie z informacjami na temat każdego zamówienia danego klienta

```
BEGIN
RETURN QUERY
SELECT
  z.id_zamowienia,
  z.kwota,
  z.pesel_klienta,
  z.vin,
  z.usluganaprawy,
  z.pesel_pracownika
FROM zamowienia AS z
WHERE z.pesel_klienta = pesel;
END;
```

---

- `sumazarobkow(p_placowkaid INT, DEFAULT NULL)` – funkcja która zwraca sumę zarobków wszystkich pracowników w danej placówce. Jeżeli funkcja zostanie wywołana bez parametru, wówczas zwrócona zostanie informacja na temat zarobków wszystkich pracowników w całej firmie

```
DECLARE
suma NUMERIC := 0;
BEGIN
IF p_placowkaid IS NOT NULL THEN
SELECT SUM(zarobki) INTO suma
FROM pracownicy
WHERE placowkaid = p_placowkaid;
ELSE
SELECT SUM(zarobki) INTO suma
FROM pracownicy;
END IF;
RETURN COALESCE(suma, 0);
END;
```

---

## 5. Koncepcja dostępu zdalnego do bazy danych

### Założenia:

- Dostęp z poziomu GUI do bazy danych odbywa się za pomocą jednego pliku który realizuje zdalne połączenie.
  - Interfejs aplikacji webowej jest intuicyjny i czytelny.
- Możliwość dodawania, usuwania i edytowania rekordów w tabelach.
  - Możliwość przeglądania statystyk i historii.

### **Planowane funkcjonalności:**

- **Pobieranie danych:** Wyświetlanie tabel (np. samochody, klienci) w interaktywnych widokach na stronie.
- **Manipulacja danymi:** Możliwość dodawania, edycji oraz usuwania rekordów w każdej tabeli z wykorzystaniem formularzy modalnych za pomocą przycisków.
- **Historia:** Możliwość wyświetlenia za pomocą przycisku historii zamówień klientów oraz wyświetlenia historii aut podwykonawcy w nowej tabeli.
- **Statystyki:** Wyświetlanie statystyk takich jak: liczba pracowników w danej placówce, suma zarobków w danej placówce, suma zamówień na auta z usługą naprawy, statystyki podwykonawcy wyświetlające sumę wartości naprawionych aut oraz współczynnik tej wartości do liczby aut.

### **Opis:**

Zdalny dostęp do bazy odbywa się za pomocą webowej aplikacji w języku PHP.

Początkowe okno startu służące do wyboru tabeli którą chcemy wyświetlić.

# Witaj w Projekcie Bazy Danych

Wybierz jedną z tabel do zarządzania:

- Samochody
- Pracownicy
- Placówki
- Stanowiska
- Klienci
- Firmy
- Podwykonawcy
- Zamowienia
- samochody\_podwykonawcy

Wyświetlana tabela z bazy danych z przyciskami do odpowiednich funkcjonalności.

## Samochody

Dodaj samochód

VIN	Marka	Model	Rocznik	Silnik	Skrzynia	Stan	Cena	Placówka ID	Akcje
1HGCM82633A123456	Honda	Accord	2010	2.4 Benzyna	Automatyczna	Nowy	35000	2	<div>Edytuj</div> <div>Usuń</div>
WDBUF56X19A876543	Mercedes	E-Class	2015	3.0 Diesel	Automatyczna	Nowy	50000	7	<div>Edytuj</div> <div>Usuń</div>
WAUZZZ8K1AA123456	Audi	A4	2018	2.0 Benzyna	Manualna	Używany	72000	0	<div>Edytuj</div> <div>Usuń</div>
VF1BMBC0548123456	Renault	Megane	2012	1.5 Diesel	Manualna	Używany	28000	7	<div>Edytuj</div> <div>Usuń</div>

Modal do dodawania nowego rekordu z warunkiem wypełnienia wszystkich pól.

## Dodaj samochód

VIN:  
2HCCM82633A158256

Marka:  
Opel

Model:  
Corsa

Rocznik:

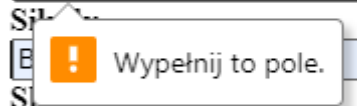
Siła:

Styl:  
Automat

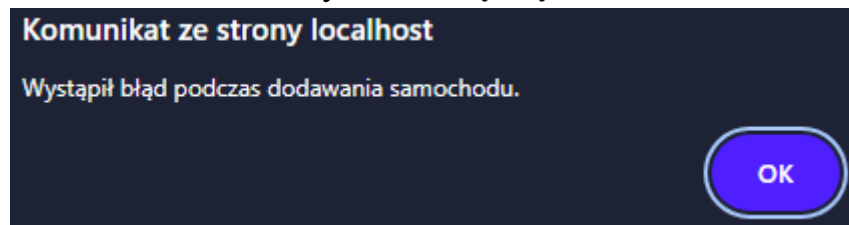
Stan:  
Nowy

Cena:  
40000.00

Placówka ID:  
2



W przypadku próby dodania rekordu a kluczu głównym który już istnieje wyświetli się błąd.



Modal przy edycji aut jest automatycznie uzupełniony oraz wyświetla nieedytowalny klucz główny rekordu na samej górze.



## Edytuj Samochód

VIN (do edycji):

VIN:

Marka:

Model:

Rocznik:

Silnik:

Skrzynia:

Stan:

Cena:

Placówka ID:

Podczas usuwania wyświetla się modal pytający o potwierdzenie usunięcia rekordu.

## Usuń samochód

Czy na pewno chcesz usunąć samochód o VIN: 1HGCM82633A123456 ?

W tabeli klientów po kliknięciu przycisku do historii zamówień wyświetlana jest na dole tabela która aktualizuje się na bieżąco przy wybraniu historii innego klienta.

97090990123	Julia	Kowalczyk		508901234	<a href="#">Edytuj</a> <a href="#">Usuń</a> <a href="#">Zobacz zamówienia</a>
98010101234	Łukasz	Kaczmarek		509012345	<a href="#">Edytuj</a> <a href="#">Usuń</a> <a href="#">Zobacz zamówienia</a>

**Zamówienia Klienta**

ID Zamówienia	Kwota	Pesel Klienta	VIN	Usługa Naprawy	Pesel Pracownika
25	172000	98010101234	JN1AZ4EH9AM123456	Tak	92030267890
<a href="#">Zamknij</a>					

W tabeli podwykonawców po pokazaniu historii wyświetla się tabela pokazująca auta nad jakimi wybrany pracodawca pracował lub pracuje.

5566778899	MotorCare	Naprawa skrzyń biegów	<a href="#">Edytuj</a> <a href="#">Usuń</a> <a href="#">Pokaż historię</a> <a href="#">Pokaż statystyki</a>
9876543210	FixCars	Naprawa układów hamulcowych	<a href="#">Edytuj</a> <a href="#">Usuń</a> <a href="#">Pokaż historię</a> <a href="#">Pokaż statystyki</a>

**Historia Podwykonawcy**

VIN	NIP Podwykonawcy
1FMCU9J96EUD23456	9876543210
WDBUF82J56X123456	9876543210
<a href="#">Zamknij</a>	

Przy wyświetlaniu statystyk danego podwykonawcy pokazuje się na dole tabela z sumą wartości aut naprawionych oraz z współczynnikiem dzielącym tą sumę przez liczbę aut.

5566778899	MotorCare	Naprawa skrzyń biegów	<a href="#">Edytuj</a> <a href="#">Usuń</a> <a href="#">Pokaż historię</a> <a href="#">Pokaż statystyki</a>
9876543210	FixCars	Naprawa układów hamulcowych	<a href="#">Edytuj</a> <a href="#">Usuń</a> <a href="#">Pokaż historię</a> <a href="#">Pokaż statystyki</a>

**Statystyki Podwykonawcy**

NIP	Suma Wartości	Wskaźnik Efektywności
9876543210	207000	103500.00
<a href="#">Zamknij</a>		

W tabeli pracowników mamy wprowadzoną funkcjonalność do zliczania zarobków w wybranej placówce.

Najpierw wybieramy jedną placówkę z listy.

**Pracownicy**

[Dodaj pracownika](#)

Wybierz placówkę:

2 - Rzeszow

-- Wybierz placówkę --

2 - Rzeszow

7 - Krakow

0 - Rzeszow

1 - Wrocław

Oblicz sumę zarobków

Pesel	Imie	Stanowiskoid	Placowkaid	Zarobki	Akcje
90010112345	Anna	1	0	4000	<a href="#">Edytuj</a> <a href="#">Usuń</a>
89021254321	Jan	15	0	6000	<a href="#">Edytuj</a> <a href="#">Usuń</a>

Następnie po kliknięciu oblicz sumę zarobków wyświetla się pod spodem wynik.

## Pracownicy

Dodaj pracownika

Wybierz placówkę: 2 - Rzeszow

Oblicz sumę zarobków

Suma zarobków: 58200

Aдекватnie w tabeli Placówek jest funkcjonalność która po wyborze placówki wyświetla ilość stanowisk w niej się znajdujących.

## Placówki

Dodaj stanowisko

Wybierz placówkę: 2 - Rzeszow

Oblicz ilość pracowników

Ilość stanowisk: 9

Tabela zamówień posiada funkcjonalność zliczającą sumę kwot wszystkich zamówień które posiadają usługę naprawy.

## Zamowienia

Dodaj zamówienie

Zlicz kwotę zamówień z naprawą

Całkowita kwota zamówień naprawy: 562000 PLN

Id_zamowienia	Kwota	Pesel klienta	VIN	Usługa naprawy	Pesel pracownika	Akcje
22	73000	87010112345	1FMCU9J96EUD23456	t	96081187654	<div>Edytuj</div> <div>Usuń</div>
23	25000	93060667890	JHLRD77804C123456	f	86051865432	<div>Edytuj</div> <div>Usuń</div>
24	28000	89050556789	VF3LCBHZ6EW123456	t	86051865432	<div>Edytuj</div> <div>Usuń</div>
25	172000	98010101234	JN1AZ4EH9AM123456	t	92030267890	<div>Edytuj</div> <div>Usuń</div>
26	18000	97090990123	3VWFA81H7WM123456	f	96081187654	<div>Edytuj</div> <div>Usuń</div>
27	51000	92020223456	1GNEK13ZX3R123456	t	94061410987	<div>Edytuj</div> <div>Usuń</div>
28	130000	94080889012	4T1BE32KX2U123456	f	88040398765	<div>Edytuj</div> <div>Usuń</div>
29	42000	89050556789	ZAR93200001512345	t	88040398765	<div>Edytuj</div> <div>Usuń</div>
30	66000	88030334567	5YJ3E1EA1HF123456	f	96081187654	<div>Edytuj</div> <div>Usuń</div>
31	38000	95040445678	KMHCM3AC4BU123456	t	86051865432	<div>Edytuj</div> <div>Usuń</div>
32	158000	92020223456	WDBUF82J56X123456	t	94061410987	<div>Edytuj</div> <div>Usuń</div>
34	72000	96070778901	WAUZZZ8K1AA123456	f	88040398765	<div>Edytuj</div> <div>Usuń</div>

Jeżeli edytujemy rekordy i zmienimy usługę naprawy wynik od razu się zmieni po następnym kliknięciu przycisku.

Przykład po edycji 3 i 4 rekordu.

## Zamowienia

Całkowita kwota zamówień naprawy: 362000 PLN

Id_zamowienia	Kwota	Pesel klienta	VIN	Usługa naprawy	Pesel pracownika	Akcje
22	73000	87010112345	1FMCU9J96EUD23456	t	96081187654	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
23	25000	93060667890	JHLRD77804C123456	f	86051865432	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
24	28000	89050556789	VF3LCBHZ6EW123456	f	86051865432	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
25	172000	98010101234	JN1AZ4EH9AM123456	f	92030267890	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
26	18000	97090990123	3VWFA81H7WM123456	f	96081187654	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
27	51000	92020223456	1GNEK13ZX3R123456	t	94061410987	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
28	130000	94080889012	4T1BE32KX2U123456	f	88040398765	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
29	42000	89050556789	ZAR93200001512345	t	88040398765	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
30	66000	88030334567	5YJ3E1EA1HF123456	f	96081187654	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
31	38000	95040445678	KMHCM3AC4BU123456	t	86051865432	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
32	158000	92020223456	WDBUF82J56X123456	t	94061410987	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
34	72000	96070778901	WAUZZZ8K1AA123456	f	88040398765	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>

Dzięki triggerom odpowiadającym za ilość liczby samochodów i miejsca pozostałego w placówkach po dodaniu bądź usunięciu samochodu tabela placówek automatycznie się aktualizuje.

Przed.

## Placówki

**Wybierz placówkę:**

Id_placowki	Miasto	Ulica	Numer budynku	Ilosc samochodow	Ilosc miejsca pozostalego	Akcje
0	Rzeszow	Hetmanska	12	6	0	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
1	Wroclaw	Wolności	12	5	15	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
2	Rzeszow	Kosynierów	15	4	16	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>
7	Krakow	Szkolna	2	7	13	<input type="button" value="Edytuj"/> <input type="button" value="Usuń"/>

# Dodaj samochód

VIN:

4F1WF5EK5B1235467

Marka:

Opel

Model:

Corsa

Rocznik:

2005

Silnik:

Benzyna

Skrzynia:

Manualna

Stan:

Używany

Cena:

12000

Placówka ID:

2

Dodaj

Anuluj

4F1WF5EK5B1235467	Opel	Corsa	2005	Benzyna	Manualna	Używany	12000	2	<div>Edytuj</div> <div>Usuń</div>
-------------------	------	-------	------	---------	----------	---------	-------	---	-----------------------------------

Po.

## Placówki

Dodaj stanowisko

Wybierz placówkę: -- Wybierz placówkę -- ▾

Oblicz ilość pracowników

Id_placowki	Miasto	Ulica	Numer budynku	Ilosc samochodow	Ilosc miejsca pozostalego	Akcje
0	Rzeszow	Hetmanska	12	6	0	<div>Edytuj</div> <div>Usuń</div>
1	Wroclaw	Wolności	12	5	15	<div>Edytuj</div> <div>Usuń</div>
2	Rzeszow	Kosynierów	15	5	15	<div>Edytuj</div> <div>Usuń</div>
7	Krakow	Szkolna	2	7	13	<div>Edytuj</div> <div>Usuń</div>

## 6. Transponowanie bazy danych na model nierelacyjny (MongoDB)

- **Analiza wymagań**

Aby przenieść bazę danych z modelu relacyjnego na MongoDB, należy dokonać następujących kroków:

- Zrozumienie struktury danych w modelu relacyjnym, w tym tabel, relacji, kluczy głównych i obcych.
  - Zidentyfikowanie danych, które mogą być przechowywane w jednej kolekcji w MongoDB jako zagnieżdżone dokumenty.
  - Zoptymalizowanie struktury dla zapytań typowych dla aplikacji.
-

## • Działania konieczne do migracji

### 1. Transformacja tabel na kolekcje:

- Relacyjne tabele, takie jak Klienci, Zamówienia, Placówki, Podwykonawcy, i Pojazdy, powinny stać się kolekcjami MongoDB.
- Powiązania typu "jeden do wielu" mogą być reprezentowane jako tablice zagnieżdżonych dokumentów w jednej kolekcji (np. zamówienia klienta w jednym dokumencie klienta).

### Przykład transformacji tabeli „Klienci” i powiązanych zamówień:

Tabela relacyjna:

- Klienci: pesel, imie, nazwisko, telefon.
- Zamówienia: id\_zamowienia, kwota, pesel\_klienta.

Kolekcja MongoDB:

```
{
  "pesel": "12345678901",
  "imie": "Jan",
  "nazwisko": "Kowalski",
  "nip": null,
  "telefon": "123456789",
  "zamowienia": [
    {
      "id_zamowienia": "Z001",
      "kwota": 2000,
      "vin": "ABC123",
      "usluga_naprawy": true,
      "pesel_pracownika": "98765432109"
    },
    {
      "id_zamowienia": "Z002",
      "kwota": 1500,
      "vin": "XYZ456",

```

```
"usluga_naprawy": false,  
"pesel_pracownika": "98765432109"  
}  
]  
}
```

---

## 2. Optymalizacja relacji typu "wiele do wielu":

Relacje tego typu (np. Podwykonawcy i Pojazdy) można realizować przez odrębne kolekcje, gdzie każda z nich zawiera referencje do dokumentów w innych kolekcjach.

### Przykład:

Kolekcja Samochody:

```
{  
  "vin": "XYZ123",  
  "model": "Audi A4",  
  "cena": 30000,  
  "nippodwykonawcy": "PL9876543210"  
}
```

Kolekcja Podwykonawcy:

```
{  
  "nip": "PL9876543210",  
  "nazwa": "AutoSerwis",  
  "adres": "ul. Serwisowa 12, Warszawa"  
}
```

---

## 3. Migracja funkcji:

Aby przekształcić te funkcje na MongoDB, musimy uwzględnić, że MongoDB jest bazą danych typu NoSQL, więc będzie to trochę inny sposób pracy z danymi. Zamiast korzystać z tradycyjnych zapytań SQL, w MongoDB używamy metod takich jak `find()`, `aggregate()`, oraz metod do manipulacji danymi, które różnią się od tych w bazach relacyjnych.



Przykład kodu w MongoDB dla tych funkcji:

### **1. Funkcja pobierająca dane o kliencie:**

MongoDB nie potrzebuje deklaracji funkcji w taki sposób jak w SQL, ale możemy po prostu zapisać zapytanie w aplikacji lub w shellu MongoDB.

```
db.klienci.find({});
```

### **2. Historia podwykonawcy** **(historia podwykonawcy):**

MongoDB nie ma odpowiednika RETURN QUERY, ale możemy zrobić zapytanie z wykorzystaniem aggregate(), jeżeli dane są w kolekcji odpowiednio powiązane.

```
db.samochody_podwykonawcy.aggregate([
{ $match: { nippodwykonawcy: "1234567890" } },
{ $project: { vin: 1, nippodwykonawcy: 1 } }
]);
```

### **3. Obliczanie statystyk podwykonawcy** **(oblicz statystyki podwykonawcy):**

MongoDB nie wspiera bezpośrednio SUM() ani COUNT() w jednym zapytaniu, ale możemy to osiągnąć przy pomocy agregacji.

```
db.samochody_podwykonawcy.aggregate([
{ $match: { nippodwykonawcy: "1234567890" } },
{
```

```
$lookup: {
  from: "samochody",
  localField: "vin",
  foreignField: "vin",
  as: "samochody_info"
},
{ $unwind: "$samochody_info" },
{
  $group: {
    _id: "$nippodwykonawcy",
    suma_wartosci_aut: { $sum: "$samochody_info.cena" },
    srednia_wartosc_auta: { $avg: "$samochody_info.cena" }
  }
}
];
```

#### **4. Liczba pracowników w placówce** **(ilość pracowników):**

MongoDB pozwala na liczenie dokumentów przy pomocy countDocuments().

Przykład:

```
db.pracownicy.countDocuments({ placowkaid: 1 });
```

Jeżeli nie podamy placowkaid, po prostu zliczymy wszystkich pracowników:

```
db.pracownicy.countDocuments({});
```

#### **5. Suma kwoty zamówień z usługą naprawy** **(suma kwoty zamowien usluga naprawy):**

MongoDB obsługuje sumowanie wartości w agregacjach, tak jak w SQL.

```
db.zamowienia.aggregate([  
  { $match: { usluganaprawy: true } },  
  { $group: { _id: null, suma: { $sum: "$kwota" } } }  
]);
```

#### **6. Zamówienia danego klienta** **(zamowienia danego klienta):**

MongoDB używa find() lub aggregate(). Przykład prostego zapytania:

```
db.zamowienia.find({ pesel_klienta: "12345678901" });
```

## **7. Suma zarobków pracowników w placówce** **(sumazarobkow):**

Podobnie jak w poprzednich przypadkach, MongoDB obsługuje sumowanie w `aggregate()`:

```
db.pracownicy.aggregate([
  { $match: { placowkaid: 1 } },
  { $group: { _id: null, suma_zarobkow: { $sum:
"$zarobki" } } }
]);
```

Jeżeli chcesz uzyskać zarobki wszystkich pracowników w firmie (bez parametru), po prostu usuń filtr:

```
db.pracownicy.aggregate([
  { $group: { _id: null, suma_zarobkow: { $sum:
"$zarobki" } } }
]);
```

---

## **4. Migracja procedur**

W MongoDB nie używamy tradycyjnych procedur jak w bazach relacyjnych, ponieważ MongoDB korzysta z innych mechanizmów do zarządzania danymi, takich jak zapytania i operacje na dokumentach. Poniżej znajdziesz odpowiedniki Twoich procedur w MongoDB:

### **1. Dodaj (Insert)**

W MongoDB operacja dodawania dokumentów do kolekcji odbywa się za pomocą metody `insertOne()` lub `insertMany()` (w zależności od tego, czy dodajesz jeden dokument, czy kilka).

```
async function dodajFirma(db, p_nazwaFirma, p_nip) {
  await db.collection('firma').insertOne({
    nazwaFirma: p_nazwaFirma,
    nip: p_nip
  });
}
```

## **2. Edytuj (Update)**

W MongoDB używamy metody updateOne() lub updateMany() do edytowania istniejących dokumentów. Możemy użyć operatora \$set do zaktualizowania wybranych pól.

```
async function edytujFirma(db, p_nazwaFirmaDoZmiany,
p_nazwaFirma, p_nip) {
  await db.collection('firma').updateOne(
    { nazwaFirma: p_nazwaFirmaDoZmiany }, // filtr
    { $set: { nazwaFirma: p_nazwaFirma, nip: p_nip } } //
zmiana wartości
  );
}
```

## **3. Usuń (Delete)**

Operacja usuwania dokumentów w MongoDB odbywa się za pomocą metody deleteOne() lub deleteMany(). Usuwamy dokumenty na podstawie określonego filtru.

```
async function usunFirma(db, p_nazwaFirma) {
  await db.collection('firma').deleteOne({ nazwaFirma:
p_nazwaFirma });
}
```

---

## 5. Migracja Triggerów

### 1. Inkrementacja liczby samochodów (dodanie samochodu)

Aby zaimplementować to w MongoDB, możemy wykorzystać **change stream**.

Za każdym razem, gdy dodamy samochód, MongoDB będzie reagować na tę zmianę i zwiększać liczbę samochodów w danej placówce .

```
const { MongoClient } = require('mongodb');

async function incrementCarCount(db) {
  const changeStream =
    db.collection('samochody').watch([
      { $match: { operationType: "insert" } }
    ]);

  changeStream.on('change', async (change) => {
    const placowkaId = change.fullDocument.placowkaId;

    await db.collection('placowki').updateOne(
      { id_placowki: placowkaId },
      { $inc: { iloscSamochodow: 1 } }
    );
  });
}
```

### 2. Dekrementacja liczby samochodów (usunięcie samochodu)

Podobnie jak w przypadku dodania samochodu, po usunięciu samochodu musimy zmniejszyć liczbę samochodów w danej placówce.

```
async function decrementCarCount(db) {
  const changeStream =
    db.collection('samochody').watch([
      { $match: { operationType: "delete" } }
    ]);

  changeStream.on('change', async (change) => {
    const placowkaId = change.documentKey._id; // ID
    usuniętego samochodu

    await db.collection('placowki').updateOne(
      { id_placowki: placowkaId },
      { $inc: { iloscSamochodow: -1 } }
    );
  });
}
```

```
);  
});  
}
```

### **3. Inkrementacja miejsca pozostałego (usunięcie samochodu)**

Po usunięciu samochodu w placówce zwiększamy liczbę dostępnych miejsc. Zamiast triggera, możemy także użyć change stream i aktualizacji po usunięciu.

```
async function incrementRemainingPlaces(db) {  
  const changeStream =  
    db.collection('samochody').watch([  
      { $match: { operationType: "delete" } }  
    ]);  
  
  changeStream.on('change', async (change) => {  
    const placowkaId = change.documentKey._id;  
  
    await db.collection('placowki').updateOne(  
      { id_placowki: placowkaId },  
      { $inc: { ilosc_miejsc_pozostalego: 1 } }  
    );  
  });  
}
```

### **4. Dekrementacja miejsca pozostałego (dodanie samochodu)**

Podobnie jak powyżej, po dodaniu samochodu zmniejszamy liczbę dostępnych miejsc w placówce.

KopiujeEdytuj

```
async function decrementRemainingPlaces(db) {  
  const changeStream =  
    db.collection('samochody').watch([  
      { $match: { operationType: "insert" } }  
    ]);  
  
  changeStream.on('change', async (change) => {  
    const placowkaId = change.fullDocument.placowkaId;  
  
    await db.collection('placowki').updateOne(  
      { id_placowki: placowkaId },  
      { $inc: { ilosc_miejsc_pozostalego: -1 } }  
    );  
  });  
}
```

```
);  
});  
}
```

## **5. Sprawdzenie liczby miejsc przed dodaniem samochodu**

W MongoDB nie ma bezpośredniego sposobu na blokowanie operacji przed ich wykonaniem. Możemy jednak sprawdzić dostępność miejsc przed dodaniem samochodu za pomocą prostego warunku.

```
async function checkRemainingPlacesBeforeInsert(db,  
placowkaId) {  
  const placowka = await  
  db.collection('placowki').findOne({ id_placowki:  
  placowkaId });  
  
  if (placowka.ilosc_miejsc_pozostalego <= 0) {  
    throw new Error('Nie można dodać samochodu: brak  
    wolnych miejsc w placówce');  
  }  
}
```

## **6. Walidacja danych**

W MongoDB walidacja danych odbywa się na poziomie schematów (JSON Schema). Należy zdefiniować schematy dla każdej kolekcji, aby zapewnić spójność danych.

### **Przykład schematu dla kolekcji Klienci:**

```
{  
  "bsonType": "object",  
  "required": ["pesel", "imie", "nazwisko", "telefon"],  
  "properties": {  
    "pesel": {  
      "bsonType": "string",
```



```
"description": "Unikalny identyfikator klienta (numer  
PESEL) "  
  
,  
  
"imie": {  
  "bsonType": "string",  
  "description": "Imię klienta"  
  
,  
  
"nazwisko": {  
  "bsonType": "string",  
  "description": "Nazwisko klienta"  
  
,  
  
"telefon": {  
  "bsonType": "string",  
  "description": "Numer telefonu klienta"  
  
,  
  
"nip": {  
  "bsonType": ["string", "null"],  
  "description": "Numer NIP klienta, jeśli dotyczy"  
}  
  
,  
  
"relationships": {  
  "zamowienia": {  
    "type": "array",  
    "description": "Lista zamówień złożonych przez  
klienta",
```

```
"items": {
  "bsonType": "object",
  "required": ["id_zamowienia", "kwota", "vin",
    "usluga_naprawy", "pesel_pracownika"],
  "properties": {
    "id_zamowienia": {
      "bsonType": "string",
      "description": "Unikalny identyfikator zamówienia"
    },
    "kwota": {
      "bsonType": "number",
      "description": "Kwota zamówienia"
    },
    "vin": {
      "bsonType": "string",
      "description": "Numer VIN pojazdu objętego
        zamówieniem"
    },
    "usluga_naprawy": {
      "bsonType": "bool",
      "description": "Flaga określająca, czy zamówienie
        dotyczy usługi naprawy"
    },
    "pesel_pracownika": {
      "bsonType": "string",
```

```
"description": "PESEL pracownika realizującego  
zamówienie"
```

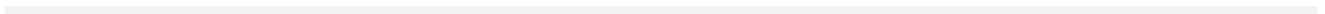
```
}
```

```
}
```

```
}
```

```
}
```

```
}
```



## 7. Obsługa indeksów:

Należy utworzyć indeksy dla często używanych pól (np. pesel, id) w celu poprawy wydajności zapytań. Indeksy pozwalają na szybsze przeszukiwanie tych pól bez potrzeby pełnego skanowania dokumentów.

```
const { MongoClient } = require('mongodb');

MongoClient.connect('mongodb://localhost:27017', {
  useUnifiedTopology: true })
  .then(client => {
    const db = client.db('mojaBaza');
    const collection = db.collection('klienci');

    return collection.createIndex({ pesel: 1 });
```