**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.

Obraz zawierający tekst, diagram, Równolegle, linia

Opis wygenerowany automatycznie

### Struktura tabel:

Firmy:

nazwafirmy, VARCHAR, PK

nip, CHAR

Klienci:

Pesel, CHAR, PK

Imie, VARCHAR,

Nazwisko, VARCHAR

Nip CHAR, FK

Telefon, VARCHAR

Placowki:

Id\_placowki, INT, PK

Miasto, VARCHAR

Ulica, VARCHAR,

Numerbudynku VARCHAR

Iloscsamochodow INT

Iloscmiejscapozostalego INT

Podwykonawcy:

Nippodwykonawcy, CHAR, PK

Nazwa, VARCHAR,

Specjalizacja, VARCHAR

Pracownicy:

Pesel, CHAR, PK

Imie, VARCHAR

Naziwsko, VARCHAR

Stanowiskoid INT, FK

Placowkaid, INT, FK

Zarobki, DOUBLE PRECISION

Samochody:

Vin, VARCHAR, PK

Marka, VARCHAR

Model, VARCHAR

Rocznik, INT

Silnik, VARCHAR

Skrzynia, VARCHAR

Stan, VARCHAR

Cena, DOUBLE PRECISION

Placowkaid, INT, FK

# Samochody\_Podwykonawcy:

Vin, CHAR, PK

Nippodwykonawcy CHAR, FK

Stanowiska:

Id\_stanowiska, INT, PK

Nazwastanowiska, VARCHAR

Minzarobki, DOUBLE PRECISION,

Maxzarobki, DOUBLE PRECISION,

Doświadczenie VARCHAR

Zamowienia:

Id\_zamowienia, INT, PK

Kwota, DOUBLE PRECISION,

Pesel\_klienta, CHAR, FK

Vin, CHAR, FK

Usluganaprawy, BOOLEAN,

Pesel\_pracownika CHAR, FK

## Zawartość Folderu

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

# strona-bazy:

Aplikacja sieciowa z dostępem do bazy danych.

# Baza.sql:

Baza danych w formacie .sql.

# Dokumentacja\_Bazy\_Danych.docx

# Dokumentacja\_Bazy\_Danych.pdf

Dokumentacja do bazy danych w 2 formatach

1. **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.

1. **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 triggery. Takie podejście zapewnia przejrzystość i optymalizację działania bazy danych.

1. **Aspekt projektowy funkcjonalności bazy danych**
2. *Schematy*

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

Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie

1. *Procedury*

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

*Przykład dla tabeli „firmy”*

*Obraz zawierający zrzut ekranu, Czcionka, tekst, czarne

Opis wygenerowany automatycznie*

*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;

1. *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;

* *inkrementacjaMiejscaPozostałego*() – w momencie gdy z danej placówki usuwany jest samochód, wartość w kolumnie „iloscmiejscapozostalego” zostaje zwiększona o jeden.

BEGIN

UPDATE placowki

SET iloscmiejscapozostalego = iloscmiejscapozostalego - 1

WHERE id\_placowki = NEW.placowkaid;

RETURN NEW;

END;

* *dekrementacjaMiejscaPozostałego*() – w momencie gdy do danej placówki dodawany jest samochów, wartość w kolumnie „iloscmiejscapozostalego” jest zmniejszana o jeden.

*BEGIN*

*UPDATE placowki*

*SET iloscmiejscapozostalego = iloscmiejscapozostalego + 1*

*WHERE id\_placowki = OLD.placowkaid;*

*RETURN OLD;*

*END;*

* check\_*iloscMiejscaPozostalego*() – w momencie gdy będziemy chcieli dodać samochód do placówki w której wartość kolumny *iloscMiejscaPozostałego* będzie wynosić 0, trigger ten na to nie pozwoli.

BEGIN

IF (SELECT iloscmiejscapozostalego FROM placowki WHERE id\_placowki = NEW.placowkaid) = 0 THEN

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

END IF;

RETURN NEW;

END;

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

# *Funkcje*

* Funkcje pobierające – każda z tabel posiada oddzielną funkcje, 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;

* ilość\_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;

1. **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.

Do uruchomienia zdalnego interfejsu należy podać user = „postgres” oraz haslo = „Korniszon1”. Baza powinna mieć nazwę „baza”.

Początkowe okno startu służące do wyboru tabeli którą chcemy wyświetlić.

**Obraz zawierający tekst, zrzut ekranu, Czcionka, design

Opis wygenerowany automatycznie**

Wyświetlana tabela z bazy danych z przyciskami do odpowiednich funkcjonalności.

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Modal do dodawania nowego rekordu z warunkiem wypełnienia wszystkich pól.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

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

Obraz zawierający tekst, zrzut ekranu, Czcionka, logo

Opis wygenerowany automatycznie

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

Obraz zawierający tekst, zrzut ekranu, Czcionka, paragon

Opis wygenerowany automatycznie

Podczas usuwania wyświetla się modal pytający o ptowierdznie usunięcia rekordu.

Obraz zawierający tekst, Czcionka, zrzut ekranu, biały

Opis wygenerowany automatycznie

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

Obraz zawierający tekst, linia, Czcionka, numer

Opis wygenerowany automatycznie

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

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

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

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

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

Najpierw wybieramy jedną placówkę z listy.

Obraz zawierający tekst, Czcionka, numer, linia

Opis wygenerowany automatycznie

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

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

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

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

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

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

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

Przykład po edycji 3 i 4 rekordu.

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

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

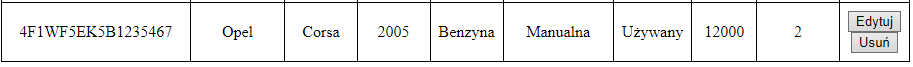
Przed.

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, Czcionka, paragon

Opis wygenerowany automatycznie



Po.

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

1. **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**

## 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"

}

]

}

## 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"

}

## 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({});

* 1. **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 } }

]);

* 1. **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" }

}

}

]);

* 1. **Liczba pracowników w placówce (ilość\_pracownikow):**

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({});

* 1. **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" } } }

]);

* 1. **Zamówienia danego klienta (zamowienia\_danego\_klienta):**

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

db.zamowienia.find({ pesel\_klienta: "12345678901" });

* 1. **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" } } }

]);

## 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\_nazwaFirmy, p\_nip) {

await db.collection('firmy').insertOne({

nazwaFirmy: p\_nazwaFirmy,

nip: p\_nip

});

}

1. **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\_nazwaFirmyDoZmiany, p\_nazwaFirmy, p\_nip) {

await db.collection('firmy').updateOne(

{ nazwaFirmy: p\_nazwaFirmyDoZmiany }, // filtr

{ $set: { nazwaFirmy: p\_nazwaFirmy, nip: p\_nip } } // zmiana wartości

);

}

1. **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\_nazwaFirmy) {

await db.collection('firmy').deleteOne({ nazwaFirmy: p\_nazwaFirmy });

}

## 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: { iloscmiejscapozostalego: 1 } }

);

});

}

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

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

KopiujEdytuj

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: { iloscmiejscapozostalego: -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.iloscmiejscapozostalego <= 0) {

throw new Error('Nie można dodać samochodu: brak wolnych miejsc w placówce');

}

}

## 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"

}

}

}

}

}

## 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 });