

**Uniwersytet Gdański**



# **SERWERY BAZY DANYCH**

**Projekt: Komis samochodowy**

**Prowadzący:**  
dr Robert Fidytek

**Wykonała:**  
Paulina Kimak  
Nr albumu 292511

**Gdańsk, 2025**

## Spis treści

1. Opis Projektu.....	3
2. Wymagania funkcjonalne systemu „Komis Samochodowy”.....	3
1. Zarządzanie komisem i placami.....	3
2. Zarządzanie samochodami.....	3
3. Rejestrowanie dostaw.....	3
4. Zarządzanie sprzedawcami.....	3
5. Zarządzanie klientami.....	4
6. Obsługa faktur.....	4
7. Kartoteka transakcji.....	4
8. Historia zmian cen.....	4
9. Zarządzanie ubezpieczeniami.....	4
10. Raportowanie i analityka.....	4
11. Bezpieczeństwo i integralność danych.....	4
12. Dodatkowe wymagania.....	5
1. Schemat ERD bazy danych dla komis samochodowego.....	5
3. Przykłady funkcji.....	6
1. Użycie funkcji – 1.wiek samochodu.....	6
2. Użycie funkcji - 2. srednia cena.....	6
3. Użycie funkcji –3.dostępny do sprzedaży.....	7
4. Użycie funkcji –4. raport klienta.....	8
5. Użycie funkcji – 5. aktualizuj cene.....	9
4. Wyzwalacze.....	10
1. Trigger 1 - Historia zmian ceny samochodu.....	10
2. Trigger 2 - Sprawdzenie gotowości samochodu do sprzedaży.....	11
3. Trigger 3- Automatyczne ustawienie samochodu jako gotowego po dostawie.....	11
4. Trigger 4- Blokowanie usunięcia samochodu powiązanego z transakcją.....	12
5. Trigger 5 - Automatyczne ustawienie daty końca polisy ubezpieczeniowej.....	12

# 1. Opis Projektu

Celem projektu jest stworzenie systemu informatycznego wspierającego pracę komis samochodowego. System umożliwia kompleksowe zarządzanie pojazdami, klientami, sprzedawcami, fakturami, transakcjami oraz historią cen i ubezpieczeniami. Projekt zakłada stworzenie relacyjnej bazy danych, która zapewni integralność danych, automatyzację procesów oraz możliwość generowania raportów sprzedaży i historii pojazdów.

## 2. Wymagania funkcjonalne systemu „Komis Samochodowy”:

### 1. Zarządzanie komisem i placami

1.1 Dodawanie, edycja i usuwanie komisów.

Dane: NIP, nazwa, państwo, miasto, ulica, nr budynku, nr lokalu, kod pocztowy, telefon, e-mail.

Walidacje:

- NIP – unikalny, 10 cyfr, brak duplikatów.
- Kod pocztowy – format „XX-XXX”.
- E-mail – poprawny format.

1.2 Dodawanie, edycja i usuwanie placów komis.

Dane: kraj, miejscowość, ulica, nr działki, przypisanie do komis.

- Każdy plac należy do jednego komis.
- Możliwość powiązania samochodów i sprzedawców z danym placem.

### 2. Zarządzanie samochodami

2.1 Dodawanie, edycja i usuwanie samochodów.

Dane: numer rejestracyjny, numer VIN, marka, model, rocznik, przebieg, pojemność silnika, rodzaj paliwa, moc, kolor, rodzaj pojazdu, ładowność, cena, opis, status „gotowy do sprzedaży”.

Walidacje:

- Numer VIN – 17 znaków, unikalny.
  - Numer rejestracyjny – unikalny, 7 znaków.
  - Cena i przebieg – wartości dodatnie.
- 2.2 Wyszukiwanie samochodów po marce, modelu, VIN, numerze rejestracyjnym, roczniku lub placu.

2.3 Zmiana statusu samochodu (np. dostępny, sprzedany, rezerwacja).

### 3. Rejestrowanie dostaw

3.1 Dodawanie nowych dostaw samochodów.

Dane: data dostawy, kraj pochodzenia, status rejestracji, stan techniczny (uszkodzony/nieuszkodzony), przypisanie do samochodu i placu.

3.2 Możliwość przeglądania historii dostaw wg daty, kraju lub pojazdu.

3.3 Automatyczne przypisywanie samochodu do placu po dostawie.

### 4. Zarządzanie sprzedawcami

4.1 Dodawanie, edycja i usuwanie sprzedawców.

Dane: imię, nazwisko, numer telefonu, e-mail, przypisanie do komis i placu.

4.2 Wyszukiwanie sprzedawców wg imienia, nazwiska lub placu.

4.3 Historia przypisań sprzedawców do placów.

## **5. Zarządzanie klientami**

5.1 Dodawanie nowych klientów indywidualnych.

Dane: imię, nazwisko, PESEL, rodzaj dokumentu, numer dokumentu, dane adresowe, telefon.

Walidacje:

- PESEL – 11 cyfr, unikalny.
- Numer dokumentu – wymagany.

5.2 Edycja danych klienta z zachowaniem unikalności numeru PESEL.

5.3 Możliwość wyszukiwania klientów po nazwisku, PESEL lub numerze dokumentu.

5.4 Archiwizacja (deaktywacja) klientów zamiast usuwania.

## **6. Obsługa faktur**

6.1 Dodawanie i edycja faktur.

Dane: numer faktury (unikalny), rabat, sposób zapłaty, status zapłaty.

6.2 Możliwość powiązania faktury z konkretną transakcją.

6.3 Walidacja unikalności numeru faktury.

6.4 Generowanie raportów faktur wg okresu lub statusu zapłaty.

## **7. Kartoteka transakcji**

7.1 Rejestrowanie transakcji sprzedaży, zakupu lub wymiany pojazdu.

Dane: rodzaj transakcji, data, uwagi, powiązania z klientem, sprzedawcą, samochodem, fakturą i placem.

7.2 Wyszukiwanie i filtrowanie transakcji po dacie, kliencie, sprzedawcy lub pojeździe.

7.3 Raporty sprzedażowe (np. miesięczna liczba transakcji, suma wartości sprzedaży, ranking sprzedawców).

## **8. Historia zmian cen**

8.1 Automatyczne zapisywanie historii zmian ceny każdego samochodu.

Dane: stara cena, nowa cena, data zmiany.

8.2 Możliwość generowania raportu zmian cen w określonym okresie.

## **9. Zarządzanie ubezpieczeniami**

9.1 Rejestrowanie polis ubezpieczeniowych dla pojazdów.

Dane: firma ubezpieczeniowa, numer polisy (unikalny), data początku i końca ubezpieczenia, koszt.

9.2 Wyszukiwanie polis po numerze, dacie końca lub firmie.

9.3 System powiadomień o kończących się polisach (np. 30 dni przed wygaśnięciem).

## **10. Raportowanie i analityka**

10.1 Raport sprzedaży: liczba i wartość sprzedanych pojazdów w określonym przedziale czasu.

10.2 Raport dostaw: zestawienie liczby dostarczonych samochodów wg kraju pochodzenia.

10.3 Raport historii cen: analiza zmian wartości pojazdów.

10.4 Raport ubezpieczeń: lista aktywnych i wygasających polis.

## **11. Bezpieczeństwo i integralność danych**

11.1 Wszystkie klucze obce mają zdefiniowane relacje ON UPDATE/DELETE CASCADE.

11.2 System waliduje unikalność kluczowych danych (NIP, PESEL, VIN, nr rejestracyjny, nr faktury, nr polisy).

11.3 Automatyczne zapisywanie dat zmian i logów operacji (np. zmiany cen, statusu sprzedaży).

## 12. Dodatkowe wymagania

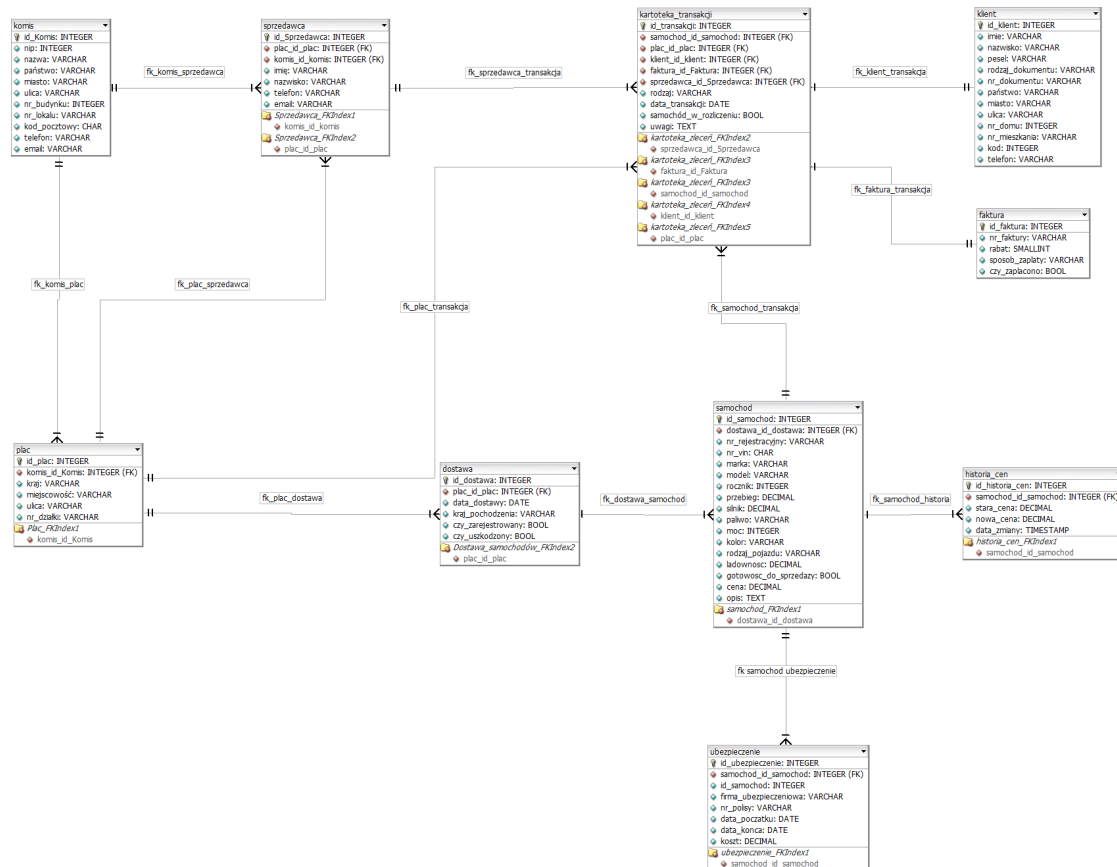
12.1 Interfejs użytkownika umożliwiający intuicyjną obsługę (np. panel zarządzania placami, samochodami, klientami).

12.2 Eksport raportów do formatu PDF, CSV lub XLSX.

12.3 Możliwość tworzenia kopii zapasowych bazy danych.

12.4 Wersjonowanie danych (np. historia zmian cen i transakcji).

## 1. Schemat ERD bazy danych dla komis samochodowego



### 3. Przykłady funkcji

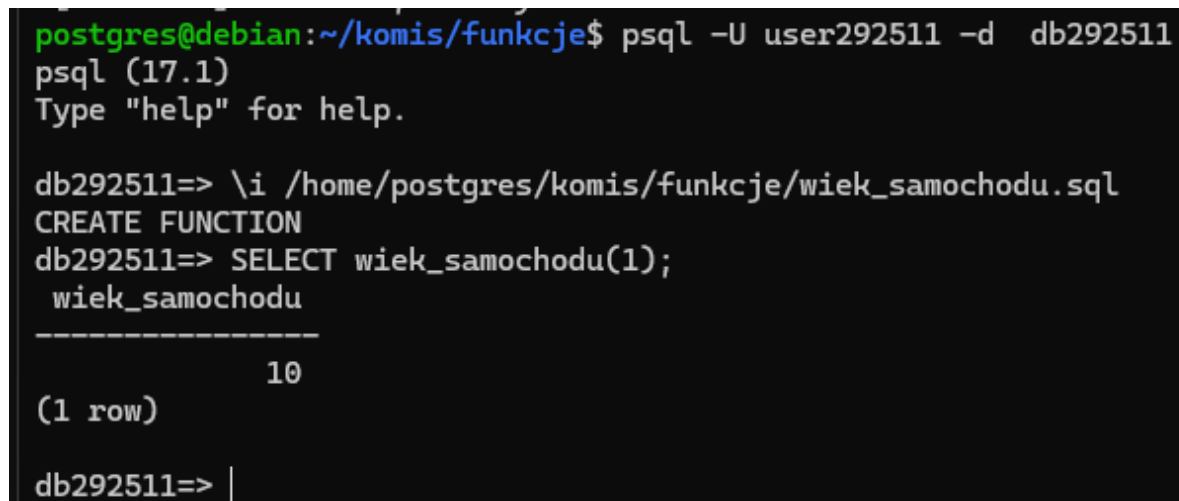
#### 1. Użycie funkcji – 1.wiek samochodu

**OPIS:** Funkcja obliczająca wiek samochodu

```
CREATE OR REPLACE FUNCTION wiek_samochodu(p_id_samochod INT)
RETURNS INT AS $$
DECLARE
    v_rocznik INT;
BEGIN
    SELECT rocznik INTO v_rocznik
    FROM samochod
    WHERE id_samochod = p_id_samochod;

    RETURN EXTRACT(YEAR FROM CURRENT_DATE) - v_rocznik;
END;
$$ LANGUAGE plpgsql;

-- Test
SELECT wiek_samochodu(1);
```



```
postgres@debian:~/komis/funkcje$ psql -U user292511 -d db292511
psql (17.1)
Type "help" for help.

db292511=> \i /home/postgres/komis/funkcje/wiek_samochodu.sql
CREATE FUNCTION
db292511=> SELECT wiek_samochodu(1);
 wiek_samochodu
-----
                10
(1 row)

db292511=> |
```

#### 2. Użycie funkcji - 2. srednia cena

**OPIS:** Funkcja obliczająca średnią cenę samochodów w danym komisie

```
CREATE OR REPLACE FUNCTION srednia_cena_komisu(p_id_komis INT)
RETURNS NUMERIC AS $$
DECLARE
    v_srednia NUMERIC;
BEGIN
    SELECT AVG(s.cena) INTO v_srednia
    FROM samochod s
```

```

JOIN dostawa d ON s.id_samochod = d.id_samochod
JOIN plac p ON d.id_plac = p.id_plac
WHERE p.id_komis = p_id_komis;

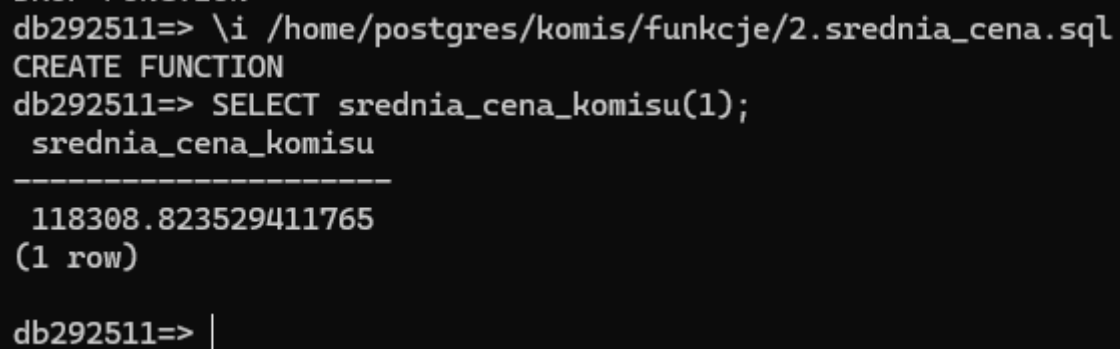
RETURN v_srednia;
END;
$$ LANGUAGE plpgsql;

```

```

-- Test
SELECT srednia_cena_komisu(1); -- średnia cena dla komisu o id_komis = 1

```



```

db292511=> \i /home/postgres/komis/funkcje/2.srednia_cena.sql
CREATE FUNCTION
db292511=> SELECT srednia_cena_komisu(1);
srednia_cena_komisu
-----
118308.823529411765
(1 row)

db292511=> |

```

### 3. Użycie funkcji –3.dostępny do sprzedaży

**OPIS:** Funkcja sprawdzająca dostępność samochodu

```

CREATE OR REPLACE FUNCTION dostepny_do_sprzedazy(p_id_samochod INT)
RETURNS BOOLEAN AS $$
DECLARE
    v_gotowy BOOLEAN;
BEGIN
    SELECT gotowy_do_sprzedazy INTO v_gotowy
    FROM samochod
    WHERE id_samochod = p_id_samochod;

    RETURN v_gotowy;
END;
$$ LANGUAGE plpgsql;

-- Test
SELECT dostepny_do_sprzedazy(5);

```

```

postgres@debian:~/komis/funkcje$ psql -U user292511 -d db292511
psql (17.1)
Type "help" for help.

db292511=> \i /home/postgres/komis/funkcje/czy_dostepny.sql
CREATE FUNCTION
db292511=> SELECT dostepny_do_sprzedazy(5);
dostepny_do_sprzedazy
-----
t
(1 row)

db292511=> SELECT dostepny_do_sprzedazy(2);
dostepny_do_sprzedazy
-----
f
(1 row)

db292511=> |

```

#### 4. Użycie funkcji –4. raport klienta

**OPIS:** Funkcja tworząca raport transakcji dla klienta

```

CREATE OR REPLACE FUNCTION raport_klienta(p_id_klient INT)
RETURNS TABLE(
    liczba_transakcji BIGINT,
    laczna_wartosc NUMERIC
) AS $$
BEGIN
    RETURN QUERY
    SELECT COUNT(k.id_transakcja), COALESCE(SUM(s.cena), 0)
    FROM kartoteka_transakcji k
    JOIN samochod s ON k.id_samochod = s.id_samochod
    WHERE k.id_klient = p_id_klient;
END;
$$ LANGUAGE plpgsql;

```

```

-- Test
SELECT * FROM raport_klienta(2);

```



```
db292511=> \i /home/postgres/komis/funkcje/4.raport_klienta.sql
CREATE FUNCTION
db292511=> SELECT * FROM raport_klienta(2);
  liczba_transakcji | łączna_wartosc
-----+-----
                10 |      1256000.00
(1 row)

db292511=> |
```

## 5. Użycie funkcji – 5. aktualizuj cene

**OPIS:** Wyświetl informacje o sprzedawcach, którzy sprzedali najwięcej samochodów

```
CREATE OR REPLACE FUNCTION aktualizuj_cene(p_id_samochod INT, p_nowa_cena NUMERIC)
RETURNS VOID AS $$
BEGIN
    INSERT INTO historia_cen(id_samochod, stara_cena, nowa_cena)
    SELECT id_samochod, cena, p_nowa_cena
    FROM samochod
    WHERE id_samochod = p_id_samochod;

    UPDATE samochod
    SET cena = p_nowa_cena
    WHERE id_samochod = p_id_samochod;
END;
$$ LANGUAGE plpgsql;

-- Test
SELECT aktualizuj_cene(3, 50000.00);
```

```
db292511=> SELECT aktualizuj_cene(3, 50000.00);
      aktualizuj_cene
-----
(1 row)

db292511=> |
```

## 4. Wyzwalacze

### 1. Trigger 1 - Historia zmian ceny samochodu

```
CREATE OR REPLACE FUNCTION log_cena_changes()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO historia_cen (id_samochod, stara_cena, nowa_cena)
    VALUES (OLD.id_samochod, OLD.cena, NEW.cena);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS log_cena_update ON samochod;
CREATE TRIGGER log_cena_update
AFTER UPDATE OF cena ON samochod
FOR EACH ROW
WHEN (OLD.cena IS DISTINCT FROM NEW.cena)
EXECUTE FUNCTION log_cena_changes();
```

```
db292511=> SELECT id_samochod, cena FROM samochod WHERE id_samochod =
1;
 id_samochod |  cena
-----+-----
          1 | 15000.00
(1 row)
```

```
db292511=> UPDATE samochod
db292511-> SET cena = cena + 5000
db292511-> WHERE id_samochod = 1;
UPDATE 1
```

```
db292511=> SELECT * FROM historia_cen WHERE id_samochod = 1 ORDER BY
data_zmiany DESC;
 id_historia | id_samochod | stara_cena | nowa_cena |      data_zmi
any
-----+-----+-----+-----+-----
          2 |          1 | 15000.00 | 20000.00 | 2025-10-24 16:4
8:58.190809
(1 row)
```

```
db292511=> |
```

## 2. Trigger 2 - Sprawdzenie gotowości samochodu do sprzedaży

```
CREATE OR REPLACE FUNCTION check_car_ready_for_sale()
RETURNS TRIGGER AS $$
BEGIN
    IF (SELECT gotowy_do_sprzedaży FROM samochod WHERE id_samochod = NEW.id_samochod)
= FALSE THEN
        RAISE EXCEPTION 'Samochód nie jest gotowy do sprzedaży';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS before_insert_kartoteka_transakcji ON kartoteka_transakcji;
CREATE TRIGGER before_insert_kartoteka_transakcji
BEFORE INSERT ON kartoteka_transakcji
FOR EACH ROW
EXECUTE FUNCTION check_car_ready_for_sale();
```

```
db292511=> INSERT INTO kartoteka_transakcji
(rodzaj, data_transakcji, samochod_w_rozliczeniu, uwagi, id_samochod,
 id_klient, id_sprzedawca, id_plac, id_faktura)
VALUES
('sprzedaż', CURRENT_DATE, false, 'Test triggera', 3, 1, 1, 1, 1);
INSERT 0 1
db292511=> |
```

## 3. Trigger 3- Automatyczne ustawienie samochodu jako gotowego po dostawie

```
CREATE OR REPLACE FUNCTION mark_car_ready_after_delivery()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE samochod
    SET gotowy_do_sprzedaży = TRUE
    WHERE id_samochod = NEW.id_samochod;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS after_insert_dostawa ON dostawa;
CREATE TRIGGER after_insert_dostawa
AFTER INSERT ON dostawa
FOR EACH ROW
EXECUTE FUNCTION mark_car_ready_after_delivery();
```

```

db292511=> INSERT INTO dostawa (data_dostawy, kraj_pochodzenia, czy_z
arejestrowany, czy_uszkodzony, id_plac, id_samochod)
VALUES (CURRENT_DATE, 'Polska', true, false, 1, 11);
INSERT 0 1
db292511=> SELECT id_samochod, gotowy_do_sprzedaży FROM samochod WHER
E id_samochod = 11;
  id_samochod | gotowy_do_sprzedaży
-----+-----
          11 | t
(1 row)

db292511=> |

```

#### 4. Trigger 4- Blokowanie usunięcia samochodu powiązanego z transakcją

```

CREATE OR REPLACE FUNCTION prevent_delete_car_in_transaction()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS(SELECT 1 FROM kartoteka_transakcji WHERE id_samochod = OLD.id_samochod)
    THEN
        RAISE EXCEPTION 'Nie można usunąć samochodu powiązanego z transakcją';
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS before_delete_samochod ON samochod;
CREATE TRIGGER before_delete_samochod
BEFORE DELETE ON samochod
FOR EACH ROW
EXECUTE FUNCTION prevent_delete_car_in_transaction();

```

```

db292511=> DELETE FROM samochod WHERE id_samochod = 2;
ERROR: Nie można usunąć samochodu powiązanego z transakcją
CONTEXT: PL/pgSQL function prevent_delete_car_in_transaction() line
4 at RAISE
db292511=> |

```

#### 5. Trigger 5 - Automatyczne ustawienie daty końca polisy ubezpieczeniowej

```

CREATE OR REPLACE FUNCTION set_default_end_date()
RETURNS TRIGGER AS $$
BEGIN

```

```

IF NEW.data_konca IS NULL THEN
    NEW.data_konca := NEW.data_poczatku + INTERVAL '1 year';
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS before_insert_ubezpieczenie ON ubezpieczenie;
CREATE TRIGGER before_insert_ubezpieczenie
BEFORE INSERT ON ubezpieczenie
FOR EACH ROW
EXECUTE FUNCTION set_default_end_date();

```

```

db292511=> INSERT INTO ubezpieczenie (id_samochod, firma_ubezpieczeni
owa, nr_polisy, data_poczatku, data_konca, koszt)
VALUES (1, 'Allianz', 'POLISA123', CURRENT_DATE, CURRENT_DATE + INTER
VAL '1 year', 1500);
INSERT 0 1
db292511=> SELECT * FROM ubezpieczenie WHERE id_samochod = 1 ORDER BY
id_ubezpieczenie DESC;
id_ubezpieczenie | id_samochod | firma_ubezpieczeniowa | nr_polisy |
data_poczatku | data_konca | koszt
-----+-----+-----+-----+-----+-----+
2025-10-24 | 1 | Allianz | POLISA123 |
2026-10-24 | 1500.00
(1 row)
db292511=> |

```