2. Baza danych służy do obsługi klientów firmy serwisującej sprzęt elektroniczny „TechFix”. **Umożliwia:**

* rejestrację zleceń serwisowych,
* przypisywanie sprzętu do zlecenia,
* rejestrowanie napraw wykonywanych przez pracowników,
* śledzenie statusu zleceń,
* generowanie raportów kosztów i historii napraw.

3. Ograniczenia danych

* NOT NULL – dla pól wymaganych (np. nazwisko klienta, data naprawy),
* CHECK – np. status zlecenia ∈ ('przyjete', 'w realizacji', 'zakonczone'),
* UNIQUE – dla emaili klientów,
* FOREIGN KEY – zapewniają spójność między tabelami,
* Typy danych – np. date, varchar(50), numeric(8,2) ograniczają format.

4. Opis perspektyw i kwerend zabezpieczających wykonywanie funkcji bazy danych

**Widok łączący dane klientów, zamówień, sprzętu i napraw:**

CREATE VIEW v\_historia\_napraw AS

SELECT k.nazwisko, z.id\_zlecenia, s.model, n.koszt, n.data\_naprawy

FROM klient k

JOIN zlecenie z ON k.id\_klienta = z.id\_klienta

JOIN sprzet s ON z.id\_zlecenia = s.id\_zlecenia

JOIN naprawa n ON s.id\_sprzetu = n.id\_sprzetu;

**Kwerendy:**

SELECT - lista zlecen w trakcie realizacji:

SELECT \* FROM zlecenie WHERE status = 'w realizacji';

INSERT - dodanie nowego klienta do bazy danych:

INSERT INTO klient (imie, nazwisko, email, telefon)

VALUES ('Anna', 'Kowalska', 'anna@example.com', '123456789');

UPDATE - zmiana statusu zlecenia na zakonczone:

UPDATE zlecenie SET status = 'zakonczone' WHERE id\_zlecenia = 101;

DELETE - usuniecie klienta o okreslonym id:

DELETE FROM klient WHERE id\_klienta = 10;

5. Transakcje i kursory

**Transakcje**

W projektowanej bazie danych transakcje są niezbędne do zapewnienia spójności i integralności danych podczas operacji przenoszenia rekordów z tabeli naprawa do tabeli drozsza\_naprawa. Cały proces (blokada rekordów, kopiowanie, usuwanie i zatwierdzenie zmian) musi zostać wykonany jako jedna atomowa jednostka, aby uniknąć sytuacji, w której np. rekord zostanie skopiowany, ale nie usunięty, lub odwrotnie.

* Transakcja gwarantuje, że jeśli w trakcie procesu wystąpi błąd lub nastąpi przerwa, cała operacja zostanie wycofana, a baza pozostanie w spójnym stanie.
* Użycie SELECT ... FOR UPDATE blokuje wybrane rekordy w tabeli naprawa z kosztami powyżej 200, zapobiegając jednoczesnym modyfikacjom przez inne transakcje, co zapewnia izolację i zapobiega konfliktom.
* Po przeniesieniu danych do nowej tabeli i usunięciu ich z oryginalnej tabeli, zatwierdzenie zmian poleceniem COMMIT zamyka operację bez ryzyka utraty danych.

Przykład transakcji:

BEGIN;

SELECT \* FROM naprawa WHERE koszt > 200 FOR UPDATE;

INSERT INTO drozsza\_naprawa(

id\_naprawy, id\_sprzetu, id\_pracownika, koszt, data\_naprawy)

SELECT

id\_naprawy, id\_sprzetu, id\_pracownika, koszt, data\_naprawy

FROM naprawa

WHERE koszt > 200;

DELETE FROM naprawa WHERE koszt > 200;

COMMIT;

VACUUM VERBOSE naprawa;

Polecenie VACUUM VERBOSE naprawa jest wykonywane po transakcji, by oczyścić i zoptymalizować tabelę naprawa po usunięciu dużej liczby rekordów.

**Kursory**

Kursory są przydatne w sytuacjach, gdy konieczne jest przetwarzanie rekordów pojedynczo — np. gdy operacje przenoszenia zależą od bardziej złożonej logiki lub gdy warunki różnią się między poszczególnymi rekordami. W powyższym przykładzie zastosowano jednak zapytania oparte na operacjach na całych zbiorach (set-based), które zazwyczaj są bardziej wydajne.

Przykład użycia kursora w języku PL/pgSQL:

DO $$

DECLARE

cur\_drozsze\_naprawy CURSOR FOR

SELECT id\_naprawy, id\_sprzetu, id\_pracownika, koszt, data\_naprawy

FROM naprawa

WHERE koszt > 200

FOR UPDATE;

v\_id\_naprawy INT;

v\_id\_sprzetu INT;

v\_id\_pracownika INT;

v\_koszt NUMERIC(10,2);

v\_data\_naprawy DATE;

BEGIN

OPEN cur\_drozsze\_naprawy;

LOOP

FETCH cur\_drozsze\_naprawy INTO v\_id\_naprawy, v\_id\_sprzetu, v\_id\_pracownika, v\_koszt, v\_data\_naprawy;

EXIT WHEN NOT FOUND;

INSERT INTO drozsza\_naprawa (id\_naprawy, id\_sprzetu, id\_pracownika, koszt, data\_naprawy)

VALUES (v\_id\_naprawy, v\_id\_sprzetu, v\_id\_pracownika, v\_koszt, v\_data\_naprawy);

DELETE FROM naprawa WHERE id\_naprawy = v\_id\_naprawy;

END LOOP;

CLOSE cur\_drozsze\_naprawy;

END;

$$ LANGUAGE plpgsql;

6. Przedstawienie możliwych do zastosowania funkcji i wyzwalaczy wraz z przykładami.

**Funkcje**

Przykładowe zastosowania:

* Obliczanie całkowitego kosztu naprawy z uwzględnieniem podatku.
* Sprawdzanie limitu kosztów naprawy i zwracanie ostrzeżeń.
* Aktualizacja statusu zlecenia po zakończeniu naprawy.

Funkcja zwracająca listę klientów z liczbą zleceń i sumą kosztów napraw

CREATE OR REPLACE FUNCTION statystyki\_klientow()

RETURNS TABLE (

id\_klienta INT,

nazwisko VARCHAR,

liczba\_zlecen BIGINT,

suma\_kosztow NUMERIC

) AS $$

BEGIN

RETURN QUERY

SELECT k.id\_klienta, k.nazwisko, COUNT(DISTINCT z.id\_zlecenia) AS liczba\_zlecen,

COALESCE(SUM(n.koszt), 0) AS suma\_kosztow

FROM klient k

LEFT JOIN zlecenie z ON k.id\_klienta = z.id\_klienta

LEFT JOIN sprzet s ON z.id\_zlecenia = s.id\_zlecenia

LEFT JOIN naprawa n ON s.id\_sprzetu = n.id\_sprzetu

GROUP BY k.id\_klienta, k.nazwisko

ORDER BY suma\_kosztow DESC;

END;

$$ LANGUAGE plpgsql;

**Wyzwalacze**

Przykładowe zastosowania:

* Automatyczna aktualizacja pola ostatnia\_aktualizacja przy każdej modyfikacji rekordu.
* Logowanie zmian w tabeli naprawa do tabeli archiwalnej.
* Sprawdzenie, czy koszt naprawy nie przekracza ustalonego limitu, i w razie potrzeby generowanie błędu lub powiadomienia.
* Automatyczne ustawianie statusu zlecenia na „Zakończone” po dodaniu ostatniej naprawy.

Wyzwalacz walidujący koszt naprawy — nie pozwala na wprowadzenie naprawy z kosztem powyżej 1000

CREATE OR REPLACE FUNCTION waliduj\_koszt\_naprawy()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.koszt > 1000 THEN

RAISE EXCEPTION 'Koszt naprawy nie może przekraczać 1000';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_waliduj\_koszt

BEFORE INSERT OR UPDATE ON naprawa

FOR EACH ROW

EXECUTE FUNCTION waliduj\_koszt\_naprawy();

INSERT INTO naprawa (id\_sprzetu, id\_pracownika, koszt, data\_naprawy)

VALUES (112, 2, 1449.63, '2024-04-14');

7. Opcjonalnie wskazanie innych przydatnych składników dla wdrożenia przedstawionej  
bazy danych

* **Zarządzanie uprawnieniami**: definiowanie ról i przydzielanie uprawnień do operacji SELECT, INSERT, UPDATE, DELETE w celu ochrony danych.

Przykład: dostęp tylko do odczytu dla analityków, pełny dostęp dla administratorów.

* **Replikacja i kopie zapasowe**: zabezpieczenie przed utratą danych, możliwość tworzenia kopii i odtwarzania bazy, replikacja może służyć do rozłożenia obciążenia na serwery.
* **Monitoring i logowanie:** monitorowanie wydajności zapytań i operacji, logowanie operacji krytycznych, np. zmian statusu zlecenia lub usuwania rekordów.
* **Dokumentacja bazy danych:** schematy, opisy tabel i kolumn, wyjaśnienie biznesowej logiki — pomaga w utrzymaniu i rozwoju bazy.

Schemat bazy:  
