Bazy danych 1, dokumentacja projektu "Dealer Samochodowy"

Glib Avrutin

2 kwietnia 2024

1 Projekt koncepcji, założenia

1.1 Zdefiniowanie tematu projektu

Projekt ”Dealer Samochodowy” zakłada rozwinięcie kompleksowego systemu wspomagającego zarządzanie działalnością dealerów samochodowych. Zastosowanie frameworka Django Python i systemu zarządzania bazą danych PostgreSQL stanowi fundament efektywnego oraz niezawodnego przechowywania danych. Głównym celem projektu jest ułatwienie intuicyjnego zarządzania informacjami dotyczącymi samochodów, klientów, transakcji i innych kluczowych aspektów funkcjonowania dealerów samochodowych.

Zgodnie z założeniami początkowymi projektu aplikacja web serwerowa Django wykonuje tylko ”surowe” zapytania do bazy danych nie używając narzędzi frameworku ułatwiających komunikację pomiędzy web serwerem a serwerem bazy danych, zatem cała logika biznesowa aplikacji jest zaimplementowana narzędziami SZBD PostgreSQL.

1.2 Analiza wymagań użytkownika

Aplikacja ”Dealer samochodowy” musi zapewniać niezawodne modyfikacje, aktualizacje oraz usuwanie rekordów, a także generowanie raportów służących do łatwiejszej analizy danych przez potencjalnych użytkowników aplikacji.

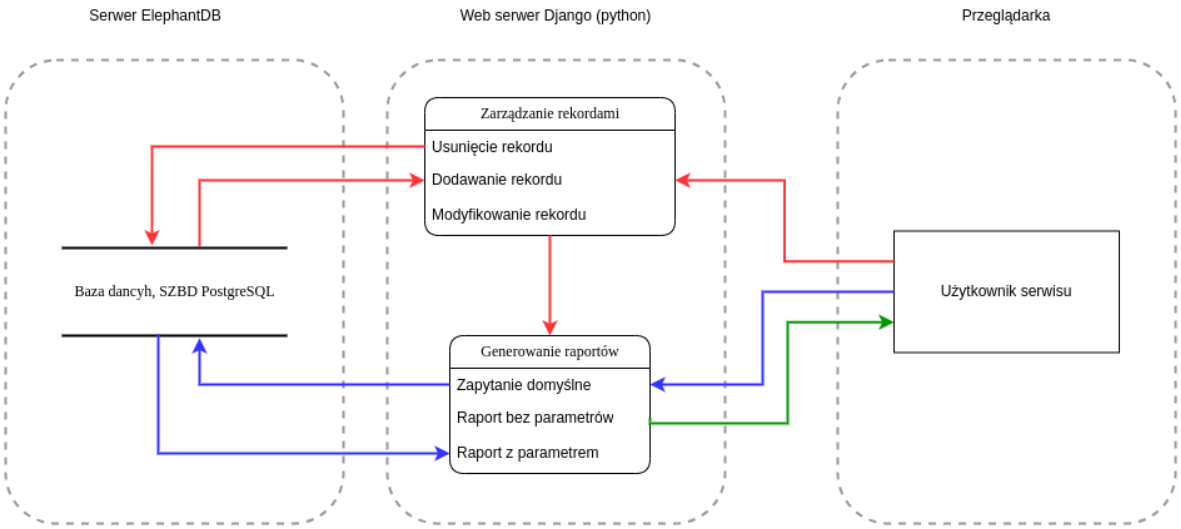
1.3 Zaprojektowanie funkcji

Podstawowe funkcje zaprojektowane w aplikacji: usunięcie i dodawanie rekordów do wszystkich tabeli (oprócz słownikowych), generowanie raportów, kontrola spójności danych za pomocą triggerów, walidacja danych wprowadzancyh przez użytkownika za pomocą procedur składowanych oraz CONSTRAINTów.

2 Projekt diagramów (konceptualny)

2.1 Budowa i analiza diagramu przepływu danych (DFD)

Do dokumentacji został dodany Diagram Przepływu Danych (DFD), który ilustruje, jak dane przemieszczają się wewnątrz systemu, ułatwiając zrozumienie struktury i interakcji między poszczególnymi elementami.



Rysunek 1: Diagram DFD aplikacji

Jak widać aplikacja web serwerowa zawiera dwie funkcje, generowanie raportów oraz zarządzanie rekordami. Funkcja generowania raportów obsługuje przesyłanie zawartości SELECTu do użytkownika, a funkcja zarządzania usuwa, dodaje ta modyfikuje rekordy. Po każdej operacji na rekordach aplikacja musi wygenerować raport żeby użytkownik zobaczył zmiany zawartości bazy.

2.2 Zdefiniowanie encji (obiektów) oraz ich atrybutów

PK - klucz główny, FK - klucz obcy.
Aplikacja zawiera następujące encje:

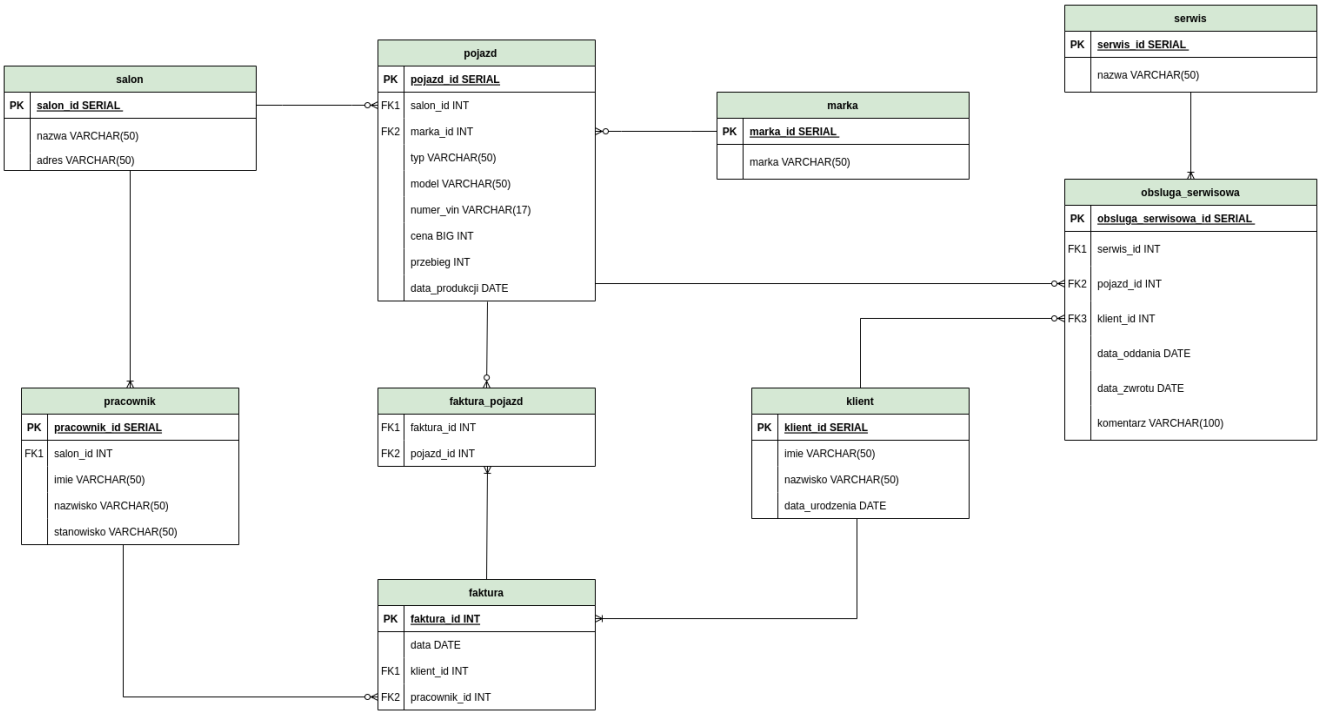
- salon
    1. `salon_id`, PK
    2. nazwa
    3. adres
- pojazd
    1. `pojazd_id`, PK
    2. `salon_id`, FK
    3. `marka_id`, FK
    4. typ
    5. model
    6. numer vin
    7. cena
    8. przebieg
    9. data produkcji

- klient
    1. `klient_id`, PK
    2. imię
    3. nazwisko
    4. data urodzenia
    5. numer telefonu
- pracownik
    1. `klient_id`, PK
    2. `salon_id`, FK
    3. imię
    4. nazwisko
    5. stanowisko
- faktura
    1. `faktura_id`, PK
    2. data
    3. `klient_id`, FK
    4. `pracownik_id`, FK
- faktura - pojazd (n do n)
    1. `faktura_id`, FK
    2. `pojazd_id`, FK
- serwis
    1. `serwis_id`, PK
    2. nazwa
- obsluga serwisowa
    1. `obsluga_serwisowa_id`, PK
    2. `serwis_id`, FK
    3. `pojazd_id`, FK
    4. `klient_id`, FK
    5. data oddania
    6. data zwrotu
    7. komentarz

- marka – encja słownikowa
    1. `marka_id`, PK
    2. marka

2.3 Zaprojektowanie relacji pomiędzy encjami (ERD)

Do powyższych encji został zaprojektowany następujący diagram ERD pokazujący relacje pomiędzy encjami w postaci graficznej.



Rysunek 2: Diagram ERD

3 Projekt logiczny

3.1 Projektowanie tabel, kluczy, indeksów

Wszystkie tabele zostały zaprojektowane zgodnie początkowymi założeniami. Cały kod opisujący encje znajduje się w pliku w `enteties.sql` załączonym do dokumentacji.

3.2 Słowniki danych

W tym projekcie został stworzony słownik marka, który zawiera wszystkie możliwe marki pojazdu. Niestety z powodu braku czasu nie udało się zrealizować wszystkie pomysły dotyczące encji słownikowych. Do tego projektu przydałyby się słowniki: typ (pojazd), model (marka), stanowisko (pracownik) i t.d. Chociaż mi się nie udało wdrążyć te funkcje do projektu, zamierzam to zrobić w przyszłości.

Zawartość słownika marka można zobaczyć w pliku `insertions.sql`. Podczas projektowania aplikacji założyłem że słownik nie może być modyfikowany przez użytkownika.

3.3 Analiza zależności funkcyjnych i normalizacja tabel

Po analizie diagramu ERD doszedłem do wniosku, że wszystkie tablice bazy danych są znormalizowane, co oznacza, że dane w systemie zorganizowane zgodnie z regułami normalizacji (3F). Skutkiem tego jest efektywność operacji zapytań, minimalizacja redundancji informacji oraz zapewnienie integralności danych.

3.4 Zaprojektowanie operacji na danych

Zgodnie z wymogami projektu zaprojektowałem widoki, SELECTy zwykłe ta z klauzulami GROUP BY, procedury składowane oraz wyzwalacze.

W przypadku widoku celą jest dostarczenie wygody pracowania z dużą ilością tabeli i zapytań z powtarzającymi się częściami kodu. Widoki w projekcie: `klient_pojazd_view`, `wartosc_faktury_view, pracownik_pojazd_view, klient_obsluga_view`.

Listę wszystkich SELECTów osadzonych w Pythonie można zobaczyć w pliku `views.py` znajdującym się w folderze `car_dealership_app` albo w pliku `selects.sql` bez jakiegokolwiek kodu Pythona. Symbol `%s` określa miejsca gdzie kursor będzie wstawiał kolejne wartości do SELECTu.

Procedury składowane w tym projekcie służą do wyrzucania wyjątków w przypadku gdy rekord nie będzie znaleziony, tym samym informując użytkownika o niemożliwości wykonania operacji. Lista procedur znajduje się w pliku `functions.sql`.

Dla tabeli `faktura_pojazd` oraz `obsluga_serwisowa` stworzyłem triggery służące do kontroli spójności danych. W przypadku `faktura_pojazd` wyzwalacz usuwa `salon_id` z tabeli samochód, a dla encji `obsluga_serwisowa` sprawdza czy samochód jest własnością klienta.

4 Projekt funkcjonalny

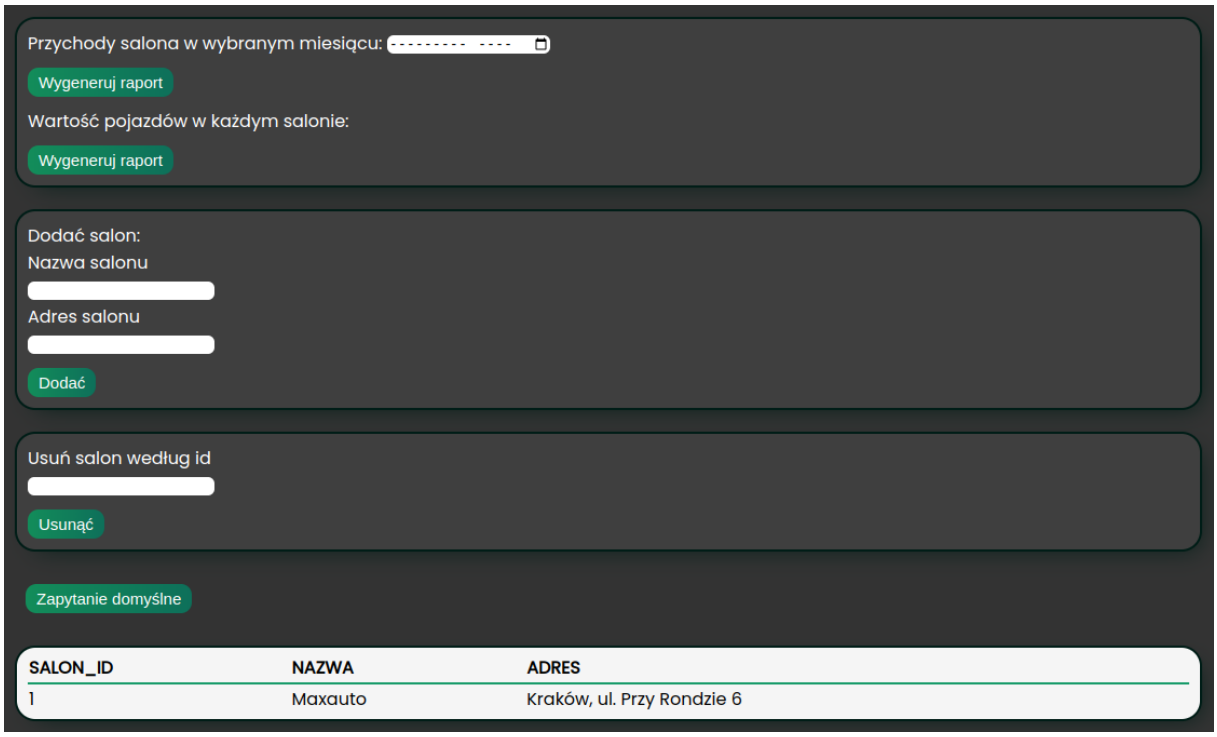
4.1 Interfejsy do prezentacji, edycji i obsługi danych:

Web aplikacja składa się z sześciu widoków, każdy z których obsługuje dodawanie, usunięcie oraz wyświetlanie rekordów.

1. Pojazdy – obsługuje tablicę pojazd
2. Salony – obsługuje tablicę salon
3. Klienci – obsługuje tablicę klient
4. Pracownicy – obsługuje tablicę pracownik
5. Faktury – obsługuje tablicy faktura oraz `faktura_pojazd`
6. Obsługa serwisowa – obsługuje tablicy serwis oraz `obsluga_serwisowa`

W każdym widoku formularzy są podzielone na trzy grupy (od góry):

1. Formularze generujące raporty
2. Formularze dodawania rekordów
3. Formularze usunięcia rekordów



Rysunek 3: Widok "Salony"

Na dole widoku znajduje się pole wyświetlające wyjątki (na obrazie nie jest widoczne), guz do generowania domyślnego raportu (Raport zawierający zawartość tabeli którą obsługuje widok) oraz tabela z rekordami.

5 Krótka dokumentacja web serwera

Web serwer został stworzony za pomocą frameworku Django w języku Python. Aplikacja django składa się z dwóch folderów, `car_dealership_app` zawierającym logikę aplikacji, `car_dealership_project` zawierającym ustawienia aplikacji oraz pliku `manage.py` uruchamiającym aplikację. Plik `urls.py` zawiera mapę ścieżek do funkcji obsługujących zapytania do serwera. Kod funkcji znajduje się w modułu `views.py`.