W relacyjnej bazie danych znajdują się trzy tabele:  
  
users //tabela z użytkownikami  
- id bigint PRIMARY KEY NOT NULL  
- nick text NOT NULL  
- login text NOT NULL  
- password text NOT NULL  
- ... // pozostałe dane o użytkowniku  
- insert\_time timestamp NOT NULL DEFAULT now()  
  
vehicles //tabela z pojazdami użytkowników  
- id bigint PRIMARY KEY NOT NULL  
- login text NOT NULL REFERENCES(users.login)  
- Brand text NOT NULL // marka pojazdu  
- model text NOT NULL // model pojazdu  
- ... // pozostałe dane o pojeździe  
- insert\_time timestamp NOT NULL  
  
insurance\_offers //tabela zawierająca oferty ubezpieczenia OC dla danego pojazdu  
- id bigint PRIMARY KEY NOT NULL  
- vehicle\_id bigint NOT NULL REFERENCES (vehicles.id)  
- insurer text NOT NULL //nazwa towarzystwa ubezpieczeniowego z którego pochodzi dana oferta  
- price float NOT NULL  
- ... // pozostałe dane o ofercie ubezpieczeniowej  
- insert\_time timestamp NOT NULL

Napisz w języku Java prosty serwer TCP/IP który otworzy Socket i zacznie nasłuchiwać przychodzących danych na wybranym porcie. Po odebraniu przychodzącego komunikatu (klient powinien wysłać id użytkownika czyli daną znajdującą się w kolumnie id tabeli "users") serwer pobierze wszystkie pojazdy i oferty ubezpieczeniowe dla danego użytkownika oraz odeśle je w odpowiedzi. Do odczytu danych z bazy użyj czystego JDBC. Użyć PostgreSQL lub innej relacyjnej bazy. Klient po odebraniu danych powinien je wypisać na konsoli w formie czytelnej dla osoby obsługującej. W ramach wykonanego zadania do podesłania SQL z utworzonej bazy, kod serwera, oraz kod klienta. Dodatkowe pytania:  
1) Co można poprawić w strukturze bazy?  
2) Jak po stronie serwera można zweryfikować czy nadawca komunikatu rzeczywiście jest właścicielem danego konta w tabeli "users"?  
3) Czy zaproponowane przez Ciebie rozwiązanie można zoptymalizować, aby pozwalało obsłużyć jednocześnie komunikaty od kilku klientów?

Odpowiedzi:

Ad 1)

1. Tabela *users*

* Dodanie do pola *id* atrybutu *auto increment* – sprawi, że wartość w tym polu będzie automatycznie inkrementowana a przez to będzie unikalna
* Dodać ograniczenie *unique* do pola login. Pozwoli uniknąć duplikatów w bazie. Dalszy rozwój aplikacji o funkcjonalność logowania byłaby wtedy możliwa na podstawie pól *login* oraz *password*
* Zmiana typu danych pola *insert\_time* z *timestamp* na *datetime*. Pole dotyczy czasu utworzenia wiersza i raczej nie powinno być automatycznie aktualizowane podczas zmian w danych któregoś z pól danego wiersza.
* Dodanie pola *update\_time* typu np. *timestamp* w celu przechowywania czasu kiedy została przeprowadzona aktualizacja danych w danym wierszu. Wartość mogła by być nullem.

1. Tabela *vehicles*

* Dodanie do pola *id* atrybutu *auto increment* – sprawi, że wartość w tym polu będzie automatycznie inkrementowana a przez to będzie unikalna
* Zmiana pola *login* jako klucz obcy do tabeli *users* na pole *id* z tabeli *users*. Łączenie tabel w relacjach jest bardziej wydajne jeśli zastosowane pola są wartościami liczbowymi a nie znakowymi
* Wydzielenie do osobnej tabeli ogólnych informacji o marce i modelu pojazdu pola *Brand, model,* dodatkowo np. pojemność silnika, typ nadwozia itp. W taki sposób przechowywany byłby w tej tabeli tylko indeks do konkretnego modelu pojazdu, jako klucz obcy. W taki sposób można uniknąć przetrzymywania nadmiarowych danych oraz ułatwiałoby modyfikację ogólnych danych dotyczących modeli samochodów
* Dodanie do pola *insert\_time* wartości domyślnej NOW(). Dodatkowo można zmienić typ danych z *timestamp* na *datetime*. Pole dotyczy czasu utworzenia wiersza i raczej nie powinno być automatycznie aktualizowane podczas zmian w danych któregoś z pól danego wiersza.
* Dodanie pola *update\_time* typu np. *timestamp* w celu przechowywania czasu kiedy została przeprowadzona aktualizacja danych w danym wierszu. Wartość mogła by być nullem.

1. Tabela *insurance\_offers*

* Dodanie do pola *id* atrybutu *auto increment* – sprawi, że wartość w tym polu będzie automatycznie inkrementowana a przez to będzie unikalna
* Wydzielenie do osobnej tabeli informacji o towarzystwach ubezpieczeniowych. Przechowywanie w tabeli *insurance\_offers* jedynie *id* towarzystwa – jako klucz obcy. Unikamy przechowywania w bazie nadmiarowych danych oraz ułatwienie podczas wprowadzania zmian w danych dotyczących towarzystw ubezpieczeniowych.
* Dodanie do pola *insert\_time* wartości domyślnej NOW(). Dodatkowo można zmienić typ danych z *timestamp* na *datetime*. Pole dotyczy czasu utworzenia wiersza i raczej nie powinno być automatycznie aktualizowane podczas zmian w danych któregoś z pól danego wiersza.
* Dodanie pola *update\_time* typu np. *timestamp* w celu przechowywania czasu kiedy została przeprowadzona aktualizacja danych w danym wierszu. Wartość mogła by być nullem.

Ad 2)

Zastosowanie mechanizmu autentykacji. Implementacja funkcjonalności wymagającej od użytkownika podania loginu i hasła do konta. Zastosowanie unikalnego loginu i weryfikacja hasła pozwoliła by na rezygnację z odpytywania użytkownika o *id* ponieważ tą wartość otrzymalibyśmy już z bazy. Hasło powinno być szyfrowane.

Ad 3)

Przedstawiona przeze mnie propozycja rozwiązania problemu opiera się na komunikacji synchronicznej pomiędzy serwerem i klientem. W celu optymalizacji należałoby zapewnić możliwość komunikacji w sposób asynchroniczny.

Moje rozwiązanie opiera się na wykonywaniu programu w jednym wątku. Tworzone są *sockety* po jednym po stronie serwera i klienta, dzięki którym możliwe jest połączenie między dwoma urządzeniami (węzłami sieci). Sockety działają na połączeniach punkt-punkt.

W przypadku komunikacji asynchronicznej dla każdego nowego połączenia tworzony jest nowy wątek, który obsługuje żądanie klienta. W przypadku serwera tworzony jest obiekt klasy *ServerSocket*, który nasłuchuje na określonym porcie. Podczas nawiązania nowego połączenia dla każdego klienta przydzielany jest obiekt klasy *Socket* – w nowym wątku. Zarządzanie *socketami* poszczególnych połączeń oraz wątkami, w których są obsługiwane odbywa się dzięki klasie *ExacutorService*.