Spis treści

1. Cel i zakres zadania

Celem projektu było zaprojektowanie rozproszonego systemu bazy danych klientów towarzystwa ubezpieczeniowego przeznaczonego do przechowywania informacji dotyczących zawieranych umów ubezpieczeniowych wraz z danymi personalnymi klientów. Baza danych wraz z aplikacją dostępową pozwoli na zapisywanie danych klientów indywidualnych i firm, danych związanych z zawartą umową ubezpieczeniową, informacji o szkodach, zdarzeniach, wypadkach zgłaszanych do towarzystwa ubezpieczeniowego oraz wyświetlanie tych danych. Baza danych wraz z aplikacją dostępową pozwala na wyświetlanie zawartości tabel, czyli danych dotyczących klientów biznesowych, czy też indywidualnych, zawartych umów, zdarzeń klienta w wypadku określonego typu ubezpieczenia. Dostęp do aplikacji będą posiadali uprawnieni użytkownicy (pracownicy towarzystwa) po wpisaniu w panelu logowania właściwego dla nich loginu i hasła.

2. Funkcje systemu

- Zapisywanie danych o nowych klientach indywidualnych i nowych klientach biznesowych
- Zapisywanie danych dotyczących zawartej polisy ubezpieczeniowej (sprzedaży)
- Zapisywanie danych o nowym rodzaju oferowanego ubezpieczenia (tylko z węzła zawierającego tabelę ubezpieczenia)

- Odczytywanie danych klientów, danych o sprzedaży i dostępnych rodzajach ubezpieczeń

3. Założenia podczas realizacji zadania

a) architektura wykorzystanej bazy danych

System rozproszony został skonfigurowany na heterogenicznej architekturze systemów operacyjnych. Wykorzystano w tym celu system Windows oraz dystrybucję systemu Linux - *Oracle Linux* udostępnioną przez firmę Oracle.
Do wykonania oraz zarządzania bazą danych wykorzystano narzędzie firmy Oracle - SQL Developer. Używane środowisko bazodanowe jest homogeniczne (Oracle 12c Release 2).

b) mechanizmy rozproszenia danych

Partycjonowanie poziome

1) tabela *klienci* - będzie partycjonowana według miejsca zamieszkania klientów (według kodu pocztowego)
2) tabela *sprzedaż* - będzie partycjonowana według miasta w którym zawarto umowę ubezpieczeniową klientów

Replikacja

Replikowana będzie tabela ubezpieczenia, zawierająca dane o rodzajach ubezpieczeń.

Alokacja danych

Tabela pracownicy będzie alokowana – dane pracowników będą przechowywane lokalnie.

c) mechanizmy wspomagające przeźroczystość

- łączniki

  Zastosowane będą dwa łączniki pozwalające na dostęp do danych między węzłami.

- synonimy

  Zastosowane będą synonimy dla tabeli klienci, sprzedaż i zdarzenia znajdujących się przeciwnym węźle.

- perspektywy

  Użyte zostaną perspektywy dla wyświetlania danych o: klientach indywidualnych, klientach biznesowych, sprzedaży dla klientów indywidualnych, sprzedaży dla klientów biznesowych.

- migawka

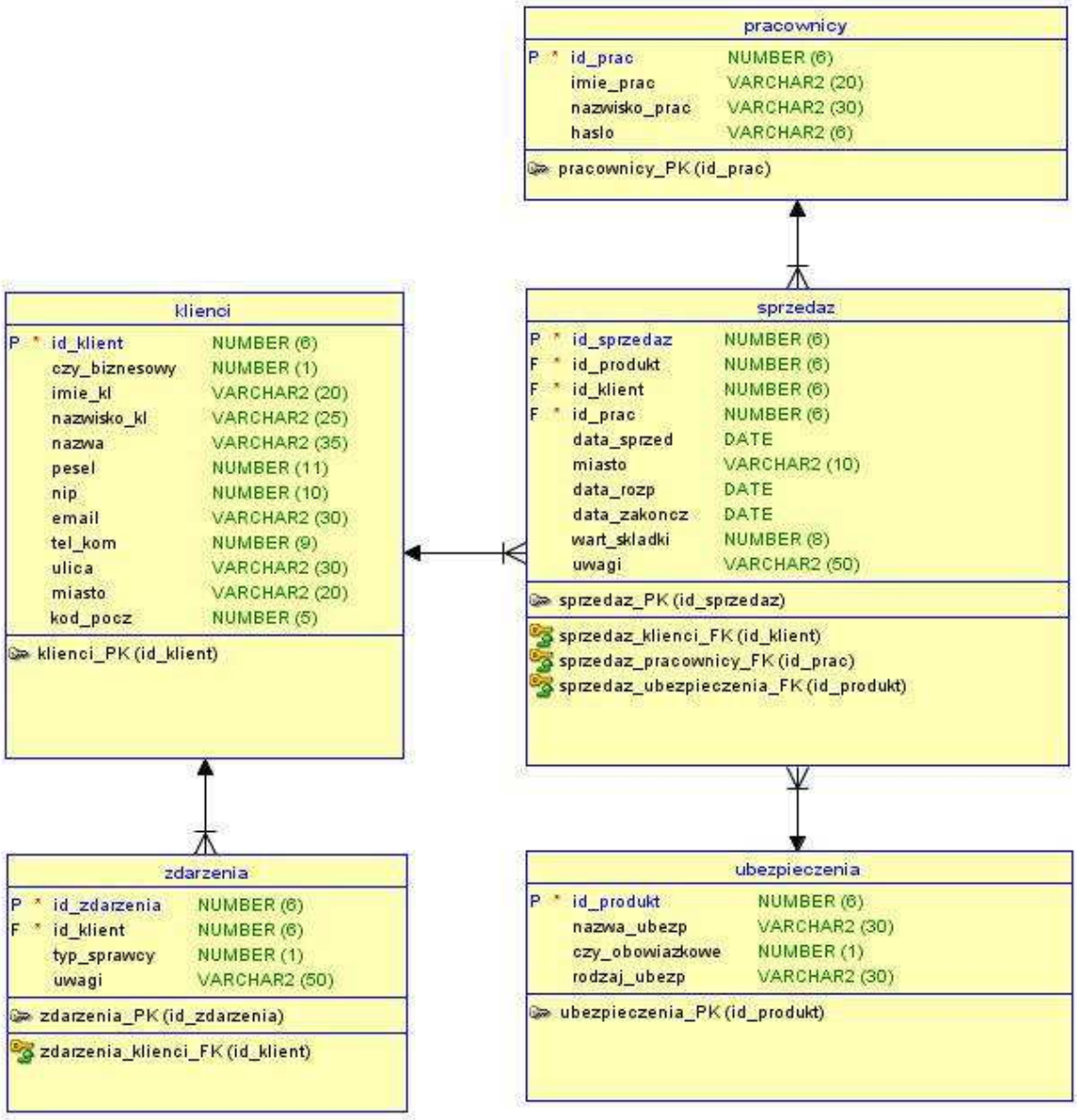
  Zostanie zastosowana migawka dla replikowania w jedną stronę danych o rodzajach ubezpieczeń.

Dostęp do aplikacji dostępowej posiadają pracownicy

4. Opis implementacji baz danych

a) struktura systemu

Rys. 1. Struktura systemu.



Rys. 2 Diagram UML tabel w bazie danych

b) przykłady implementacji wybranych elementów bazy danych

```
CREATE TABLE klienci (
    id_klient       NUMBER(6) NOT NULL,
    czy_biznesowy   NUMBER,
    imie_kl         VARCHAR2(20),
    nazwisko_kl     VARCHAR2(25),
    nazwa           VARCHAR2(35),
    pesel           NUMBER(11),
    nip             NUMBER(10),
    email           VARCHAR2(30),
    tel_kom         NUMBER(9),
    ulica           VARCHAR2(30),
    miasto          VARCHAR2(20),
    kod_pocz        NUMBER(5),

    --dodanie klucza podstawowego do tabeli klieci
  CONSTRAINT  klienci_pk PRIMARY KEY (id_klient)
);
```

Rys. 3. Przykład implementacji tabeli *klienci*.

```
CREATE TABLE sprzedaz (
    id_sprzedaz     NUMBER(6) NOT NULL,
    id_produkt      NUMBER(6) NOT NULL,
    id_klient       NUMBER(6) NOT NULL,
    id_prac         NUMBER(6) NOT NULL,
    data_sprzed     DATE,
    miasto          VARCHAR2(10),
    data_rozp       DATE,
    data_zakoncz    DATE,
    wart_skladki    NUMBER(8),
    uwagi           VARCHAR2(50),

    --dodanie klucza podstawowego do tabeli sprzedaz
    CONSTRAINT sprzedaz_pk PRIMARY KEY (id_sprzedaz),

    --dodanie klucza obcego wzglÃ™dem tabeli klieci
    CONSTRAINT sprzedaz_klienci_fk FOREIGN KEY (id_klient)
    REFERENCES klienci (id_klient),

    --dodanie klucza obcego wzglÃ™dem tabeli pracownicy
    CONSTRAINT sprzedaz_pracownicy_fk FOREIGN KEY ( id_prac )
    REFERENCES pracownicy ( id_prac ),

    --dodanie klucza obcego wzglÃ™dem tabeli ubezpieczenia
    CONSTRAINT sprzedaz_ubezpieczenia_fk FOREIGN KEY ( id_produkt )
    REFERENCES ubezpieczenia ( id_produkt )
);
```

Rys. 4. Przykład implementacji tabeli *sprzedaz*.

```
CREATE MATERIALIZED VIEW mv_ubezpieczenia
BUILD IMMEDIATE
REFRESH FORCE
START WITH SYSDATE + (1/(24*60*3))
NEXT SYSDATE + (1/(24*60*12))
AS SELECT * FROM ubezpieczenia@VM_LINK;
```

Rys. 5. Implementacja migawki wykorzystywanej do replikacji tabeli ubezpieczenia.

```
CREATE OR REPLACE VIEW wszyscy_klienci
AS SELECT * FROM klienci
UNION
SELECT * FROM klienci_poznan;

--perspektywa dla wszystkich klientÃłw indywiualych
CREATE OR REPLACE VIEW wszyscy_klienci_ind
AS SELECT id_klient, imie_kl, nazwisko_kl, pesel, email, tel_kom, ulica,
miasto, kod_pocz FROM klienci WHERE czy_biznesowy = 0
UNION
SELECT id_klient, imie_kl, nazwisko_kl, pesel, email, tel_kom, ulica,
miasto, kod_pocz FROM klienci_poznan WHERE czy_biznesowy = 0;
```

Rys. 6. Przykład implementacji wybranych perspektyw.

```
--synonim dla migawki mv_ubezpieczenia
CREATE PUBLIC SYNONYM ubezpieczenia FOR mv_ubezpieczenia;

--synonim dla klientÃłw z Poznania (vm)
CREATE PUBLIC SYNONYM klienci_poznan FOR klienci@VM_LINK;

--synonim dla sprzedaĹźy z Poznania (vm)
CREATE PUBLIC SYNONYM sprzedaz_poznan FOR sprzedaz@VM_LINK;
```

Rys. 7. Przykład utworzonych synonimów na jednej z maszyn (host).

```
--utworzenie triggera dla wstawiania danych do tabeli klienci
CREATE OR REPLACE TRIGGER dane_klienta_trg
AFTER INSERT ON klienci
FOR EACH ROW
DECLARE
wyjatek_biz      EXCEPTION;
wyjatek_imie     EXCEPTION;
wyjatek_nazwisko EXCEPTION;
wyjatek_nazwa    EXCEPTION;

BEGIN
    IF :new.kod_pocz > 59999 THEN

    IF  :new.czy_biznesowy>1
    THEN raise wyjatek_biz;
    END IF;

    --obsluga kopiowania danych klientÃ³w indywidualnych
    IF :new.czy_biznesowy=0 THEN

    IF  :new.imie_kl IS NULL
    THEN raise wyjatek_imie;
    END IF;

    IF  :new.nazwisko_kl IS NULL
    THEN raise wyjatek_nazwisko;
    END IF;

    --wstawianie danych do tabeli klienci na vm
    INSERT INTO klienci@VM_LINK(id_klient, czy_biznesowy, imie_kl, nazwisko_kl,
    pesel, email, tel_kom, ulica, miasto, kod_pocz) values (:new.id_klient,
    :new.czy_biznesowy, :new.imie_kl, :new.nazwisko_kl, :new.pesel, :new.email,
    :new.tel_kom, :new.ulica, :new.miasto, :new.kod_pocz);

    END IF;

    IF :new.czy_biznesowy=1 THEN

    IF  :new.nazwa IS NULL
    THEN raise wyjatek_nazwa;
    END IF;

    INSERT INTO klienci@VM_LINK(id_klient, czy_biznesowy, nazwa, nip, email,
    tel_kom, ulica, miasto, kod_pocz) values (:new.id_klient,
    :new.czy_biznesowy, :new.nazwa, :new.nip, :new.email, :new.tel_kom,
    :new.ulica, :new.miasto, :new.kod_pocz);

    END IF;

    DBMS_OUTPUT.PUT_LINE('Przeniesiono rekord');

    END IF;

    --obsluga wyjatku zwiazanego z nieprawidlowy, polem czy_biznesowy
    EXCEPTION
    WHEN wyjatek_biz THEN
    DBMS_OUTPUT.PUT_LINE('Blad. Nieprawidlowa wartosc pola czy_biznesowy');

    WHEN wyjatek_imie THEN
    DBMS_OUTPUT.PUT_LINE('Blad. Wartosc pola imie_kl nie moze byc pusta');

    WHEN wyjatek_nazwisko THEN
    DBMS_OUTPUT.PUT_LINE('Blad. Wartosc pola nazwisko_kl nie moze byc pusta');

    WHEN wyjatek_nazwa THEN
    DBMS_OUTPUT.PUT_LINE('Blad. Wartosc pola nazwa nie moze byc pusta');

END;
/
```

Rys. 8. Przykład wykorzystywanego triggera do wstawiania danych do tabeli “klienci” (host).

```
/**
Procedura dodaje dane klienta do tabeli klienci.
W procedurze wystÄ™puje obsluga przenoszenia danych klientÃlw miedzy wÄ™zami.
**/
CREATE OR REPLACE PROCEDURE wstaw_dane_klienta
(czy_b IN NUMBER, imie IN VARCHAR2, nazwisko IN VARCHAR2,
 nazwa IN VARCHAR2, nr_pesel IN NUMBER, nr_nip IN NUMBER, ad_email IN VARCHAR2,
 telefon IN NUMBER, ulica IN VARCHAR2, miasto IN VARCHAR2, kod_poczt IN NUMBER)
 AS
 nr_klienta   NUMBER(6);
 kod_pocztowy NUMBER(5);
 CURSOR znajdz IS SELECT id_klient, kod_pocz FROM klienci where kod_pocz>59999;
 BEGIN
--wstaw dane klienta do tabeli klineci
INSERT INTO klienci(czy_biznesowy, imie_kl, nazwisko_kl, nazwa, pesel, nip,
email, tel_kom, ulica, miasto, kod_pocz) VALUES (czy_b, imie, nazwisko, nazwa,
nr_pesel, nr_nip, ad_email, telefon, ulica, miasto, kod_poczt);


--otworz kursor i znajdz nadmiarowe dane
OPEN znajdz;
FETCH znajdz INTO nr_klienta, kod_pocztowy;
WHILE znajdz%FOUND
LOOP
--usun zduplikowane dane
DELETE FROM klienci WHERE kod_pocz = kod_pocztowy;
DBMS_OUTPUT.PUT_LINE('Usunieto rekord dla klienta o nr' || nr_klienta);
FETCH znajdz INTO nr_klienta, kod_pocztowy;
END LOOP;
COMMIT;
END;
/
```

Rys. 9. Przykład wykorzystywanej procedury do wstawiania danych do tabeli “klienci” (host).

```
--utworzenie triggera dla autoinkremencji klucza podstawowego
CREATE OR REPLACE TRIGGER klienci_trg
BEFORE INSERT ON klienci
FOR EACH ROW
DECLARE
maks_host NUMBER;
maks_vm   NUMBER;
BEGIN
    IF :new.id_klient IS NULL THEN
      SELECT MAX(id_klient) INTO maks_vm from klienci@VM_LINK;
      SELECT MAX(id_klient) INTO maks_host from klienci;

      IF maks_vm IS NOT NULL THEN
        IF maks_host < maks_vm THEN
          :new.id_klient := maks_vm+1;
        END IF;

        IF maks_host > maks_vm THEN
          :new.id_klient := maks_host+1;
        END IF;

      END IF;

      IF maks_vm IS NULL THEN
        :new.id_klient := klienci_seq.NEXTVAL;
      END IF;

    END IF;
END;
/
```

Rys. 10. Implementacja wyzwalacza (triggera) służącego do autoinkrementacji klucza podstawowego w tabeli klienci (host).

5. Sposób uruchamiania i testowania systemu bazodanowego

Pierwszym krokiem wykonanym w czasie tworzenia systemu było stworzenie połączenia między dwoma systemami komputerowymi. W tym celu na obu komputerach, w naszym wypadku jest to host (Windows) oraz maszyna wirtualna (dystrybucja Linuxa), stworzono wirtualne połączenie lokalne przy pomocy którego maszyna wirtualna może połączyć się z maszyną hosta. Najprostszym sprawdzeniem poprawnej konfiguracji było użycie komendy “ping” w terminalu oraz w konsoli cmd.

Po uzyskaniu wiedzy, iż maszyny komunikują się w sieci lokalnej rozpoczęliśmy dalsze prace. Zostały utworzone tabele na jednostce hosta oraz maszyny wirtualnej. Następnie stworzona została migawka, jak na rysunku 5. Z odpowiednim interwałem czasowym replikuje ona tabele ubezpieczenia na inny serwer baz danych (maszyna wirtualna).

Kolejnym krokiem było stworzenie perspektyw oraz wyzwalaczy (triggerów) pozwalających nam na dalsze prace z bazą danych, gdzie część z nich jest rozproszona. Po dodaniu procedur do naszego systemu bazodanowego, przetestowaliśmy je. W tym celu dodając nowe rekordy (na przykład do tabeli klienci) na obu jednostkach patrzyliśmy, czy trafiają one do odpowiednich tabel, na odpowiednich maszynach. Fragmentacja okazała się przebiegać pomyślnie.

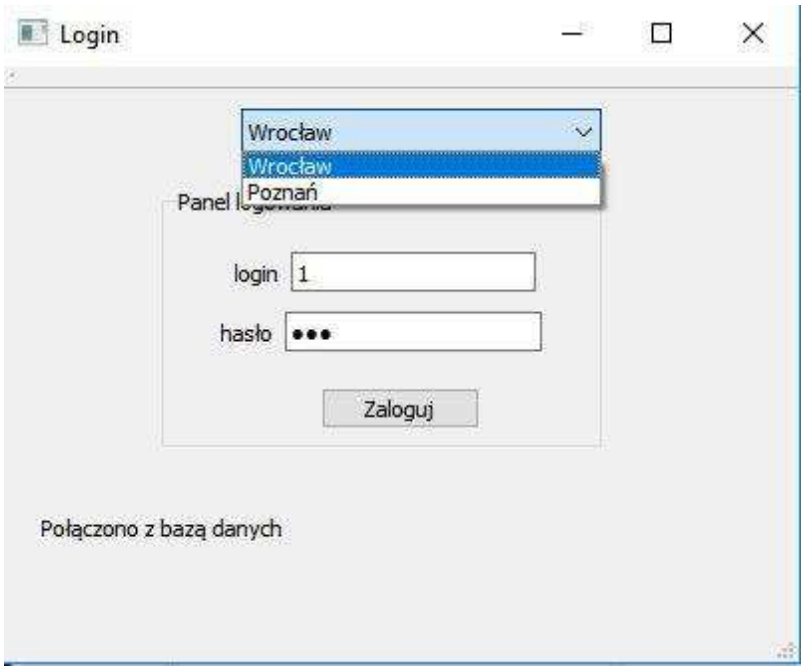
Po przetestowaniu oraz stwierdzeniu pozytywnego działania mechanizmów rozpraszania danych oraz mechanizmów przetwarzania danych została stworzona aplikacja dostępowa.

6. Opis implementacji aplikacji dostępowej

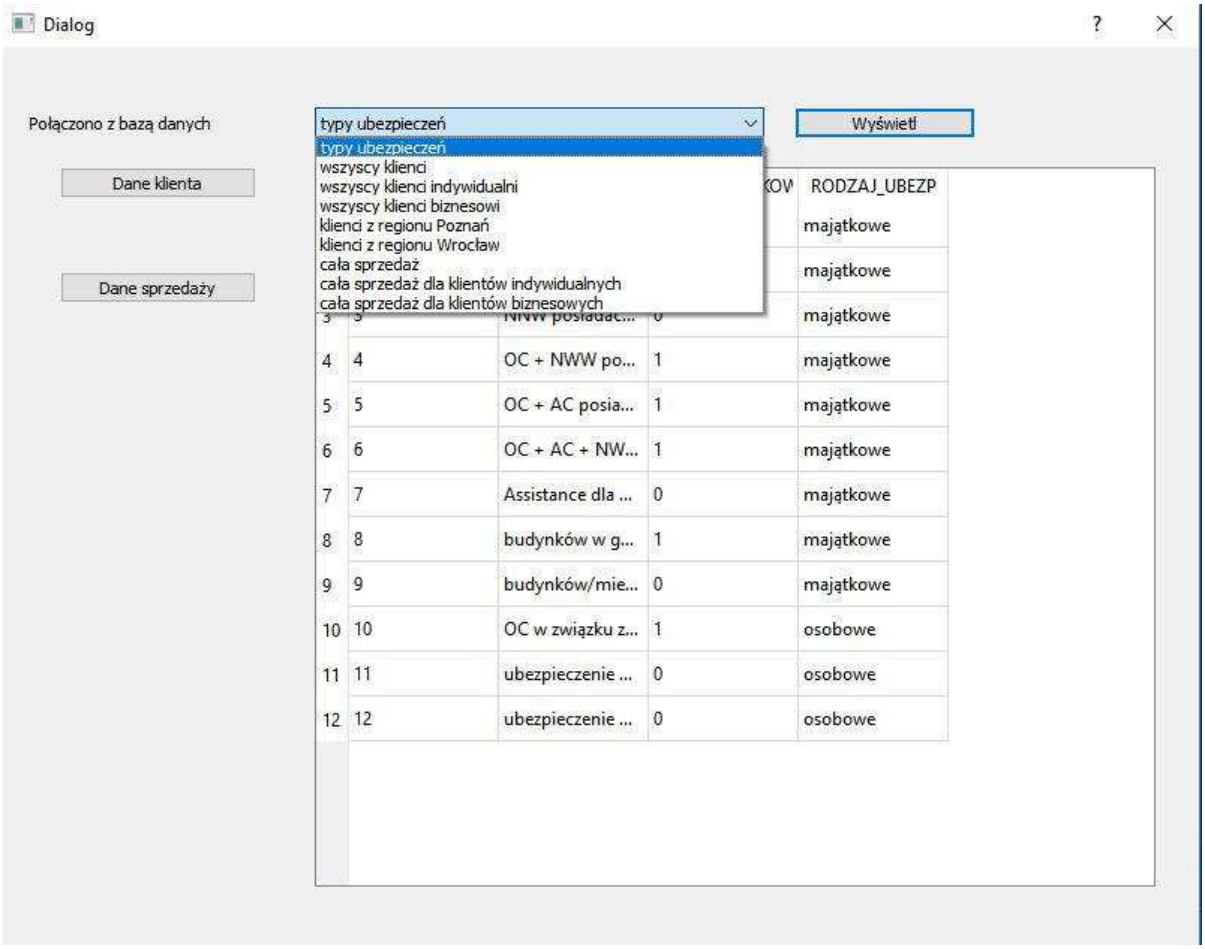
Graficzny interfejs użytkownika pozwala dla zalogowanych pracowników, którzy są użytkownikami aplikacji na dostęp do bazy danych. Pracownicy Ci, po zalogowaniu się oraz połączeniu z bazą danych mają możliwość wyświetlania rekordów znajdujących się w bazie danych. Mają możliwość wyświetlenia danych

spośród różnych tabel. Niektórymi spośród nich jest wyświetlanie typów ubezpieczeń z tabeli “ubezpieczenia”, klientów indywidualnych lub biznesowych z tabeli “klienci”.

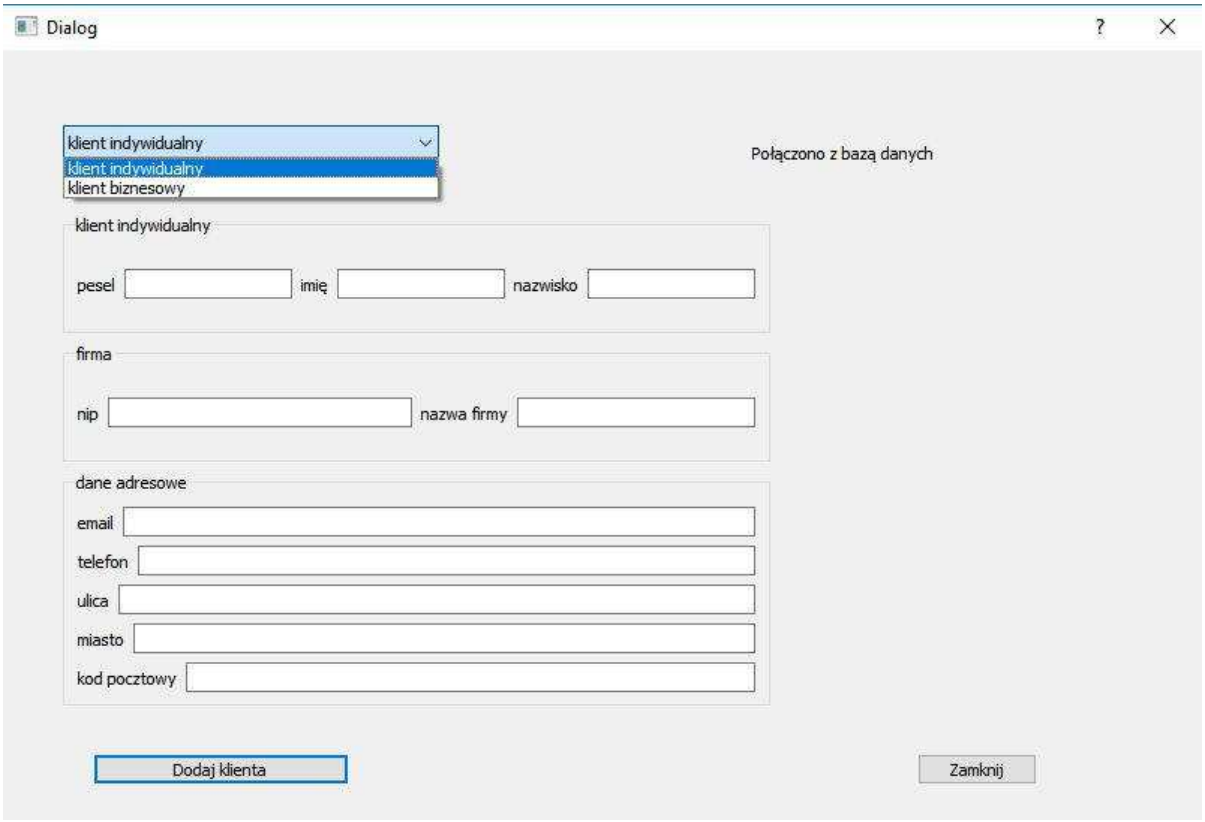
Interfejs użytkownika został stworzony w narzędziu QT Creator.



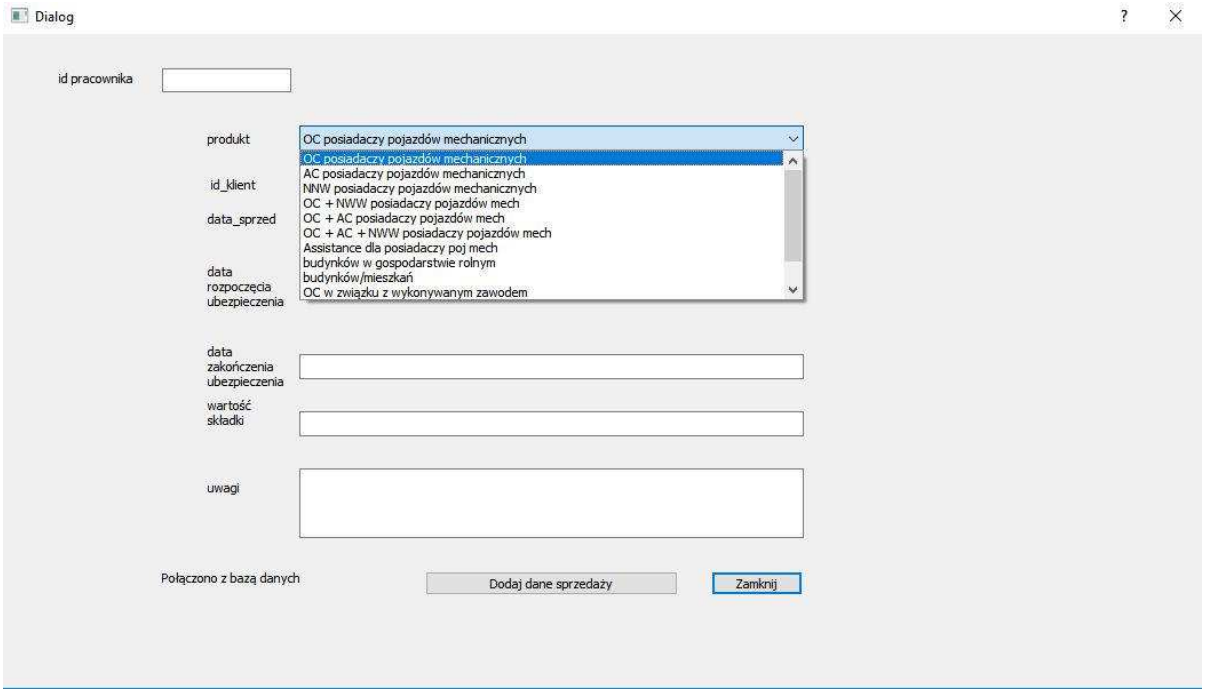
Rys. 11. Panel logowania użytkownika aplikacji dostępowej.



Rys. 12. Okno główne aplikacji dostępowej do przeglądania danych.



Rys. 13. Okno aplikacji dostępowej do dodawania danych klienta.



Rys. 14. Okno aplikacji dostępowej do dodawania danych o sprzedaży.

7. Podsumowanie

Zadaniem wykonywanym przez autorów było zaprojektowanie oraz wdrożenie systemu bazodanowego dla firmy ubezpieczeniowej. Wszelkie założenia zostały zrealizowane:

- system jest rozproszony - znajduje się w wielu instancjach,
- zaimplementowano replikację danych,
- odpowiednie tabele zostały pofragmentowane,
- aplikacja dostępowa pozwala odpowiednim użytkownikom na wyświetlanie, oraz wprowadzanie nowych rekordów do bazy danych.

Zastosowane mechanizmy rozpraszania bazy danych pozwalają na zwiększenie niezawodności aplikacji mogących korzystać z bazy danych. Replikacja kluczowych tabel pozwala na zminimalizowanie możliwości utraty danych, natomiast zastosowanie fragmentacji pozwala na szybszy dostęp do szukanych danych - nie obciążamy jednej jednostki komputerowej, lecz dwie, co zwiększa szybkość dostępową.

Zgodnie z konwencją przyjętą podczas rozpoczynania projektu, użytkownik korzystający, tudzież mający dostęp do bazy danych z zewnątrz widzi ją jako jedna instancja bazy danych. Wynika to stąd, iż całość integracji skupia się w bazie danych, a nie poza nią.

Podczas pierwszej styczności z oprogramowaniem Oracle - SQL Developer środowisko to, nie jest do końca intuicyjne, jeżeli za kryterium przyjmiemy łatwość konfiguracji nawiązania połączenia między komputerami. Z drugiej strony natomiast, wbudowane narzędzia oraz metody pozwalają na stosunkowo łatwe wdrożenie mechanizmów rozpraszania. Stworzona aplikacja dostępowa zapewnia dostęp autoryzowanym podmiotom do wyświetlania oraz manipulowania danymi.

Podczas trwania projektu, zapoznano się z podstawami rozproszonych baz danych, stosowanymi metodami ich rozpraszania.

8. Wykorzystane pozycje literaturowe

[1] Wrembel R., Bębel B.: *Oracle. Projektowanie rozproszonych baz danych*, wyd. Helion, Gliwice 2003

[2] Strona z dokumentacją baz danych Oracle, [online], dostęp: https://docs.oracle.com/en/database/oracle/oracle-database/index.html (z dnia 07.01.2019)

[3] Strona projektu Qt, [online], dostęp: https://www.qt.io (z dnia 07.01.2019)