Dokumentacja projektu

Kliniki dentystyczne

Autorzy:

Jakub Flis nr albumu: 117794

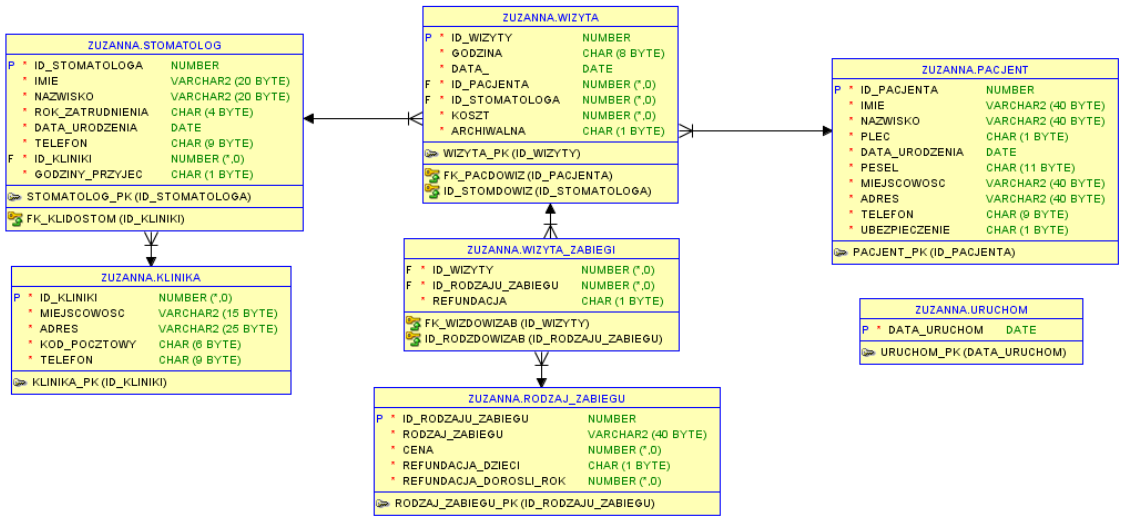
Zuzanna Heller nr albumu: 117797

Technologie: Oracle Database, Java (JDBC, JavaFX)

Problematyka i cele projektu:

Opracowanie systemu umawiania wizyt w sieci klinik dentystycznych wykorzystujące przyjazny użytkownikowi interfejs graficzny, możliwość dokonywania operacji CRUD, a także wyręczenie użytkownika w żmudnych czynnościach takich jak sprawdzanie cen zabiegów stomatologicznych oraz czy są one refundowane.

Diagram ERD:



Opis diagramu:

Baza danych posiada tabele zawierające dane klinik (KLINIKA), stomatologów (STOMATOLOG), pacjentów (PACJENT) i oferowanych zabiegów (RODZAJ_ZABIEGU).

Każdy stomatolog jest przypisany do danej kliniki, więc czerpie z niej klucz obcy.

Zapisywanie umawianych wizyt odbywa się w tabeli WIZYTA. Jest połączona bezpośrednio z pacjentem i stomatologiem, a pośrednio z kliniką i rodzajami zabiegów.

Połączenie tabeli wizyt z tabelą rodzajów zabiegów odbywa się za pośrednictwem tabeli transferowej WIZYTA_ZABIEGI. Przypisuje ona do ID_WIZYTY (klucz obcy), ID_RODZAJU_ZABIEGU (klucz obcy), co pozwala do jednej wizyty przypisać wiele zabiegów.

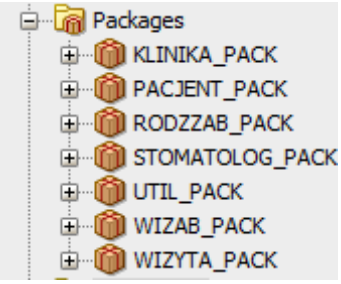
Zabiegi mogą być refundowane dla dzieci (pole REFUNDACJA_DZIECI), albo x-krotnie w ciągu roku dla dorosłych (pole REFUNDACJA_DOROSLI_ROK).

Wizyty, które już się odbyły są oznaczane jako archiwalne.

Ponadto jest też tabela URUCHOM, która przechowuje daty logowania się do programu (jedna data na dzień).

Klucze Główne są ustawiane dla nowych rekordów przy użyciu ustawienia GENERATED ON NULL AS IDENTITY – przy dodawaniu rekordu można podać id o wartości null.

Zastosowane procedury bazodanowe:



Baza danych posiada 7 pakietów z procedurami. W ich skład wchodzi 6 pakietów podstawowych, zawierających procedury CRUD dla każdej z 6 podstawowych tabeli, a także 1 pakiet z procedurami ogólnymi.

Przykład- pakiet procedur dla tabeli STOMATOLOG:

```
create or replace PACKAGE stomatolog_pack AS
    PROCEDURE dodaj (
        z_imie             IN VARCHAR2,
        z_nazwisko         IN VARCHAR2,
        z_rok_zatrudnienia IN CHAR,
        z_data_urodzenia   IN VARCHAR2,
        z_telefon          IN CHAR,
        z_id_kliniki       IN NUMBER,
        z_godziny_przyjec  IN CHAR
    );

    PROCEDURE edytuj (
        z_id_stomatologa   IN NUMBER,
        z_imie             IN VARCHAR2,
        z_nazwisko         IN VARCHAR2,
        z_rok_zatrudnienia IN CHAR,
        z_data_urodzenia   IN VARCHAR2,
        z_telefon          IN CHAR,
        z_id_kliniki       IN NUMBER,
        z_godziny_przyjec  IN CHAR
    );

    PROCEDURE usun (
        z_id_stomatologa IN NUMBER
    );

END stomatolog_pack;
```

Procedury wykonują proste polecenia dodawania, edycji i usuwania rekordów (w przypadku tabeli łączącej tylko dodawania i usuwania).

Pakiet CRUDowy zawiera sprawdzanie poprawności podanych id oraz walidację danych przy użyciu wyrażeń regularnych REGEX.

Przykład:

```
SELECT
    COUNT(1)
INTO checkid
FROM
    klinika
WHERE
    id_kliniki = z_id_kliniki;

IF checkid = 0 THEN
    raise_application_error(-20027, 'STOMATOLOG.ID_KLINIKI nie istnieje');
END IF;
```

Jeśli nie znaleziono podanego id kliniki w tabeli KLINIKA, wyrzucony zostanie błąd.

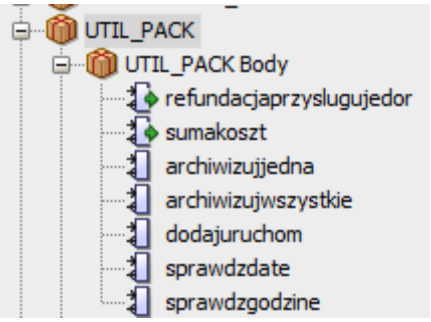
```
IF NOT regexp_like(z_nazwisko, '^[[:alpha:]]{0,20}$') THEN
    raise_application_error(-20023, 'STOMATOLOG.NAZWISKO niepoprawna wartosc');
END IF;

IF NOT regexp_like(z_rok_zatrudnienia, '^[[:digit:]]{4}$') THEN
    raise_application_error(-20024, 'STOMATOLOG.ROK_ZATRUDNIENIA niepoprawna wartosc');
END IF;

IF NOT regexp_like(z_data_urodzenia, '^[0-9]{2}-[0-9]{2}-[0-9]{4}$') THEN
    raise_application_error(-20025, 'STOMATOLOG.DATA_URODZENIA niepoprawna wartosc');
ELSE
    util_pack.sprawdzdate(z_data_urodzenia);
END IF;
```

Nazwisko zawiera maksymalnie 20 liter alfabetu, rok zatrudnienia stomatologa to maksymalnie 4 cyfry, a data urodzenia ma format 00-00-0000 do sprawdzenia czego zostaje także użyta procedura SPRAWDZDATE, która z podanego łańcucha spróbuje utworzyć datę Oracle'ową.

Gdy którykolwiek blok sprawdzający napotka na problem, wyrzuci predefiniowany błąd o zindywidualizowanym kodzie błędu.

Przykład błędów:

```
s_imie_zla_wartosc EXCEPTION;
PRAGMA exception_init ( s_imie_zla_wartosc, -20022 );
s_nazw_zla_wartosc EXCEPTION;
PRAGMA exception_init ( s_nazw_zla_wartosc, -20023 );
s_rok_zatrudnienia_zla_wartosc EXCEPTION;
PRAGMA exception_init ( s_rok_zatrudnienia_zla_wartosc, -20024 );
s_dataur_zla_wartosc EXCEPTION;
PRAGMA exception_init ( s_dataur_zla_wartosc, -20025 );
s_telefon_zla_wartosc EXCEPTION;
PRAGMA exception_init ( s_telefon_zla_wartosc, -20026 );
s_id_kliniki_zla_wartosc EXCEPTION;
PRAGMA exception_init ( s_id_kliniki_zla_wartosc, -20027 );
s_godziny_przyjec_zla_wartosc EXCEPTION;
PRAGMA exception_init ( s_godziny_przyjec_zla_wartosc, -20028 );
```



- Funkcja REFUNDACJAPRZYSLUGUJEDOR sprawdza czy maksymalna liczba bezpłatnych zabiegów danego rodzaju dla danego pacjenta została wyczerpana w tym roku; przyjmuje id pacjenta i id zabiegu, oblicza, ile razy w tym roku dany pacjent wykorzystał refundację na dany zabieg. Jeśli wykorzystał jej limit przypisany do zabiegu, zwróci 0. Jeśli refundacja przysługuje, zwróci 1.

```
FUNCTION refundacjaprzysługujedor (
    id_pac IN NUMBER,
    id_zab IN NUMBER
) RETURN NUMBER IS
    tenrok      NUMBER;
    max_rok     NUMBER;
    przysługuje NUMBER;
BEGIN
    SELECT
        COUNT(*) "refundacja"
    INTO tenrok
    FROM
             wizyta w
        INNER JOIN wizyta_zabiegi wz ON w.id_wizyty = wz.id_wizyty
    WHERE
            w.id_pacjenta = id_pac
        AND wz.id_rodzaju_zabiegu = id_zab
        AND EXTRACT(YEAR FROM w.data_) = EXTRACT(YEAR FROM sysdate)
        AND wz.refundacja = '1';

    SELECT
        refundacja_dorosli_rok
    INTO max_rok
    FROM
        rodzaj_zabiegu
    WHERE
        id_rodzaju_zabiegu = id_zab;

    IF tenrok < max_rok THEN
        przysługuje := 1;
        dbms_output.put_line('Refundacja przysługuje');
    ELSE
        przysługuje := 0;
        dbms_output.put_line('Refundacja nie przysługuje');
    END IF;

    dbms_output.put_line('Funkcja zakończona.');
    RETURN przysługuje;
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line('Wystąpił błąd');
END refundacjaprzysługujedor;
```

- Funkcja SUMAKOSZT zwraca łączny koszt danej wizyty; przyjmuje id wizyty, następnie przy pomocy kursora znajduje wszystkie zabiegi przypisane do tej wizyty i sumuje ich koszty (jeśli nie są refundowane).

```
FUNCTION sumakoszt (
    id_wiz IN NUMBER
) RETURN NUMBER IS

    suma        NUMBER;
    cenazabiegu NUMBER;
    CURSOR kursor IS
    SELECT
        *
    FROM
             wizyta w
        INNER JOIN wizyta_zabiegi wz ON w.id_wizyty = wz.id_wizyty
    WHERE
        w.id_wizyty = id_wiz;

    row         kursor%rowtype;
BEGIN
    suma := 0;
    OPEN kursor;
    LOOP
        FETCH kursor INTO row;
        EXIT WHEN kursor%notfound;
        IF row.refundacja = '0' THEN
            SELECT
                cena
            INTO cenazabiegu
            FROM
                rodzaj_zabiegu
            WHERE
                rodzaj_zabiegu.id_rodzaju_zabiegu = row.id_rodzaju_zabiegu;

            suma := suma + cenazabiegu;
        END IF;

    END LOOP;

    CLOSE kursor;
    dbms_output.put_line('Funkcja zakończona.');
    RETURN suma;
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line('Wystąpił błąd');
END sumakoszt;
```

- Procedury archiwizujące ustawiają wartość pola archiwalna przy wizytach jako 1;

„ARCHIWIZUJJEDNA” przyjmuje id danej wizyty i ją archiwizuje.

```
PROCEDURE archiwizujjedna (
    id_wiz IN NUMBER
) IS
BEGIN
    UPDATE wizyta
    SET
        archiwalna = '1'
    WHERE
        wizyta.id_wizyty = id_wiz;

    dbms_output.put_line('Procedura zakończona.');
    COMMIT;
EXCEPTION
    WHEN OTHERS THEN
        dbms_output.put_line('Wystąpił błąd');
END archiwizujjedna;
```

„ARCHIWIZUJWSZYSTKIE” archiwizuje wszystkie wizyty z datą starszą od dzisiejszej.

```
PROCEDURE archiwizujwszystkie IS

    CURSOR kursor IS
    SELECT
        *
    FROM
        wizyta
    WHERE
        data_ <= trunc(sysdate);

    row wizyta%rowtype;
BEGIN
    OPEN kursor;
    LOOP
        FETCH kursor INTO row;
        EXIT WHEN kursor%notfound;
        UPDATE wizyta
        SET
            archiwalna = '1'
        WHERE
            wizyta.id_wizyty = row.id_wizyty;

    END LOOP;

    CLOSE kursor;
    dbms_output.put_line('Archiwizacja zakończona.');
END archiwizujwszystkie;
```

- Procedura DODAJURUCHOM dodaje daną datę do tabeli uruchom; najpierw sprawdza, czy podana data już jest w tabeli. Jeśli jej nie ma, zostanie dodana.

```
PROCEDURE dodajuruchom (
    dzis_data IN DATE
) IS
    checkid NUMBER(7);
BEGIN
    SELECT
        COUNT(1)
    INTO checkid
    FROM
        uruchom
    WHERE
        data_uruchom = dzis_data;

    IF checkid = 0 THEN
        INSERT INTO uruchom VALUES ( dzis_data );

        dbms_output.put_line('Dodano dzisiejszą datę uruchomienia ' || to_char(dzis_data));
        COMMIT;
    END IF;

END dodajuruchom;
```

- Procedury SPRAWDZDATE, SPRAWDZGODZINE pomagają w walidowaniu poprawności dat i godzin wprowadzanych jako parametry procedur CRUD.

```
PROCEDURE sprawdzdate (
    data IN VARCHAR2
) IS
    dataur_zla_wartosc EXCEPTION;
    PRAGMA exception_init ( dataur_zla_wartosc, -20010 );
BEGIN
    dbms_output.put_line(to_date(data, 'DD-MM-YYYY'));
EXCEPTION
    WHEN OTHERS THEN
        raise_application_error(-20010, 'DATA_URODZENIA nie istnieje taki dzien');
END sprawdzdate;
```

```
PROCEDURE sprawdzgodzine (
    godzina IN VARCHAR2
) IS
    godzina_zla_wartosc EXCEPTION;
    PRAGMA exception_init ( godzina_zla_wartosc, -20010 );
BEGIN
    dbms_output.put_line(to_date(godzina, 'HH24:MI:SS'));
EXCEPTION
    WHEN OTHERS THEN
        raise_application_error(-20010, 'GODZINA nie istnieje taka godzina');
END sprawdzgodzine;
```

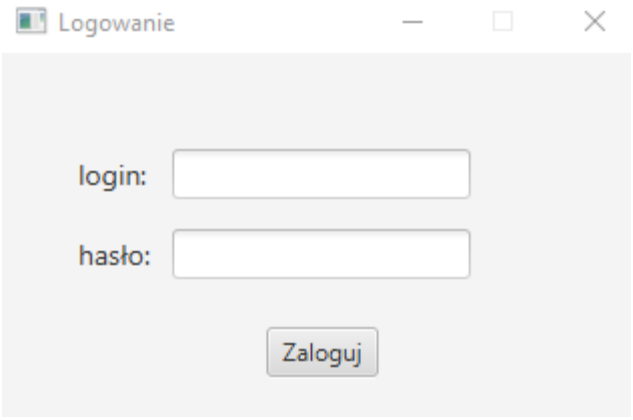
Zastosowany został też trigger ARCHIWIZUJAUTOMATYCZNIE, który jest wywoływany podczas wstawienia dzisiejszej daty do tabeli uruchom i służy do archiwizowania wszystkich wizyt starszych niż dzisiaj.

```
create or replace TRIGGER ArchiwizujAutomatycznie
    BEFORE
    INSERT
    ON uruchom
BEGIN
    dbms_output.put_line('Automatyczna archiwizacja wizyt');
    util_pack.archiwizujwszystkie;
END;
```

Proces uruchomienia aplikacji:

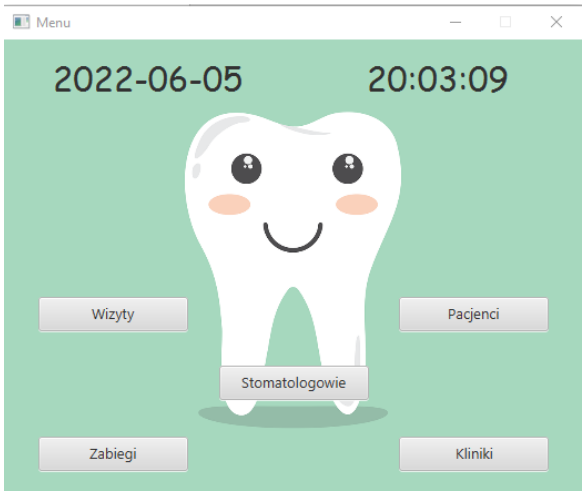
Logowanie – dane są przechowywane wewnątrz kodu źródłowego, są to dwa klucze:

„admin, admin” lub „ , ”.



Wpisuję admin, admin, loguję się. Ekran logowania znika.

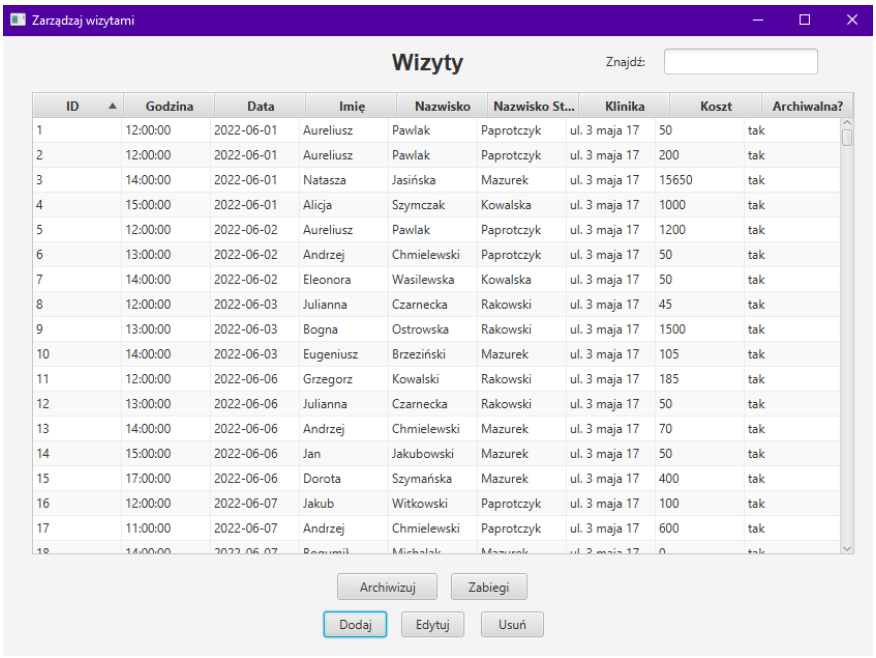
Zostaje nawiązanie połączenie z bazą danych. Program po logowaniu wywołuje procedurę DODAJURUCHOM dodającą dzisiejszą datę do tabeli URUCHOM. Dodanie dzisiejszej daty do tej tabeli wyzwala trigger ARCHIWIZUJAUTOMATYCZNIE, który archiwizuje (ustawia wartość pola ARCHIWALNA na 1) wizyty starsze niż dzisiaj.



Okno główne posiada przyciski prowadzące do pięciu podstawowych tabel bazy danych.

W górnej części ekranu jest data uruchomienia programu i aktualna godzina.

Przechodzę do tabeli wizyty.

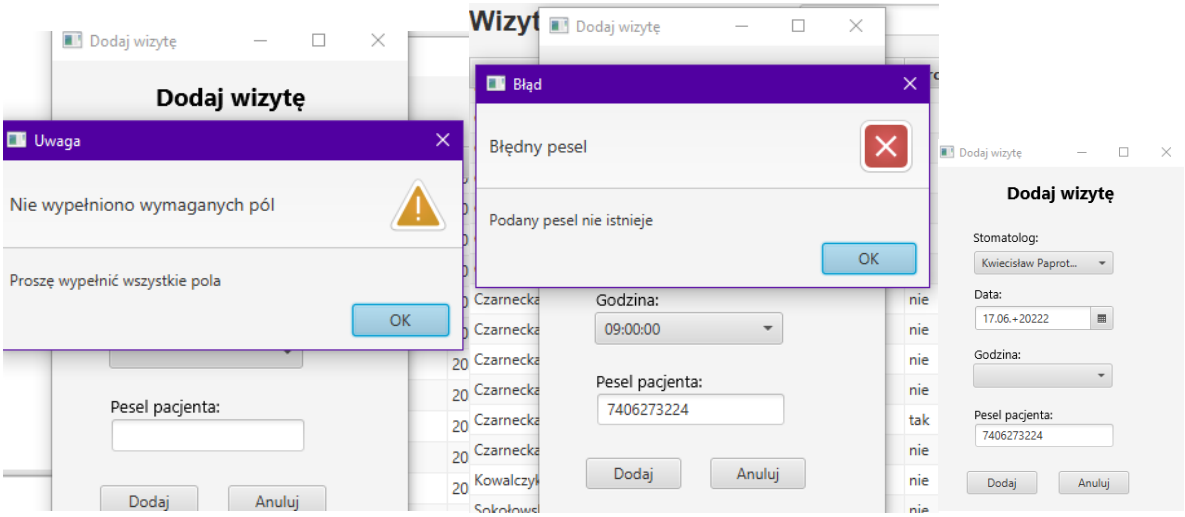


Wyświetlane są rekordy wszystkich wizyt. ID_PACJENTA został zastąpiony rubrykami imię i nazwisko (pacjenta), ID_STOMATOLOGA przez nazwisko stomatologa, a ID_KLINIKI przez jej adres.

Jest możliwość dodania nowego rekordu , a po wybraniu rekordu z tabeli także jego edycji, archiwizacji, usunięcia, czy dodania zabiegów.

Wybieram dodanie wizyty.

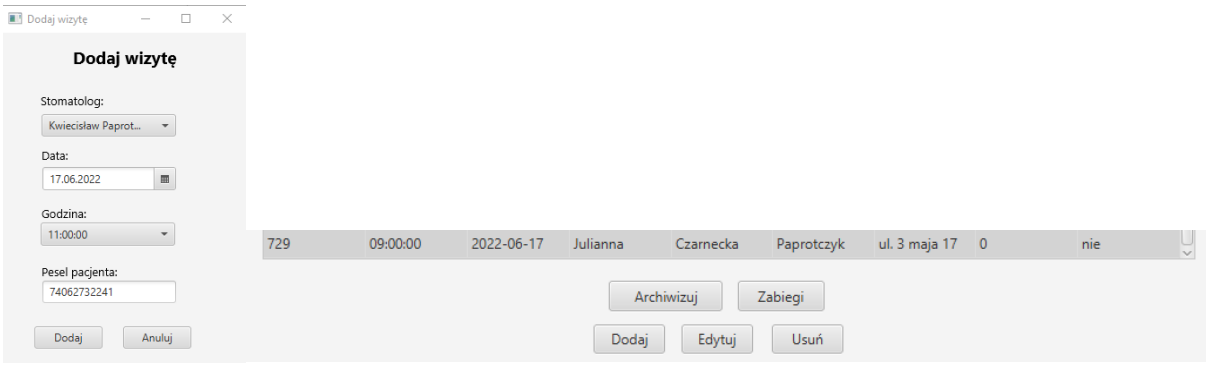
Przy niekompletnym wypełnieniu pól warstwa GUI wyświetli komunikat:



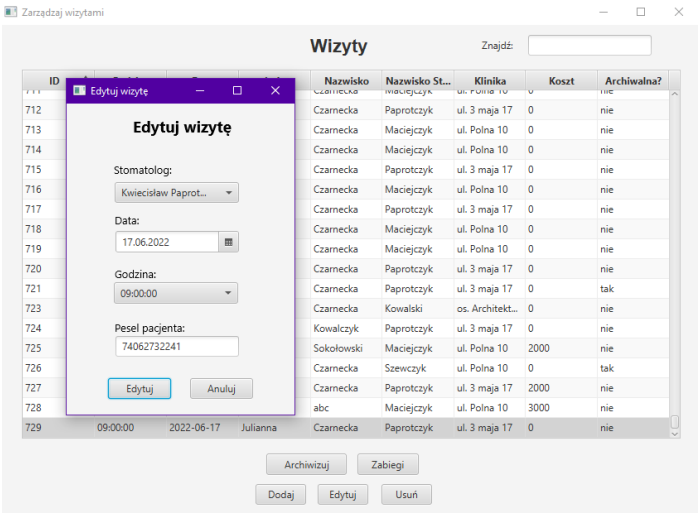
Przy dodawaniu wizyty, jeśli funkcja DatabaseHandler.DodajWizyte() wykona zapytanie sql które nie zwróci id pacjenta dla podanego peselu, to wyrzuci błąd IllegalArgumentException który zostanie wyłapany przez DodajWizyteController.

Aby wyświetlić dostępne godziny musimy najpierw wybrać stomatologa i datę- do tego czasu wybór godziny jest zablokowany. Jeśli później wprowadzimy innego stomatologa lub datę, godziny są resetowane i wybierane na podstawie nowych danych.

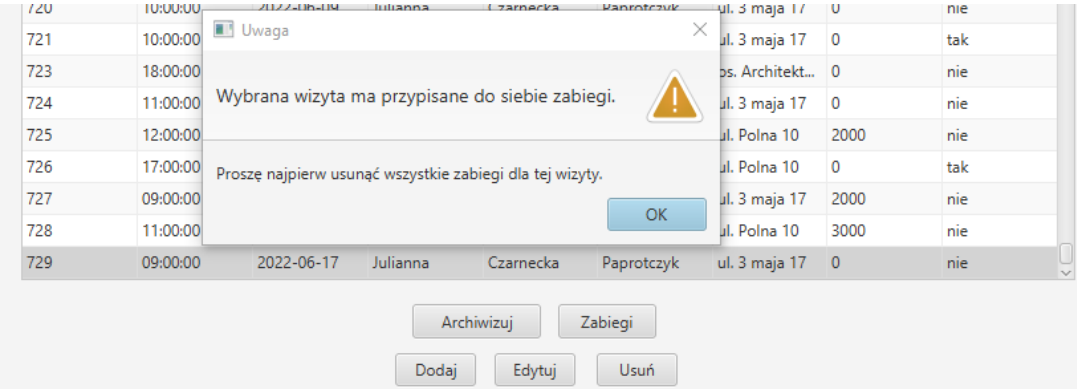
Jeżeli zostanie wprowadzona zła data (można wprowadzić ją ręcznie lub przez DatePickera) to nie wyświetlą się godziny w podanym dniu dla podanego stomatologa.



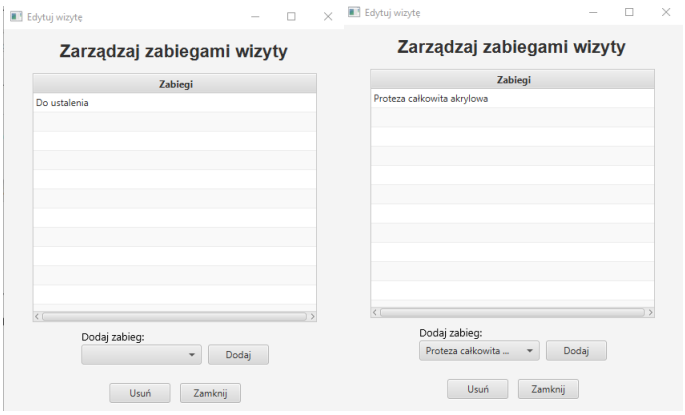
Po otworzeniu okna edycji pola zostaną automatycznie uzupełnione. Również zachodzi walidacja błędów.



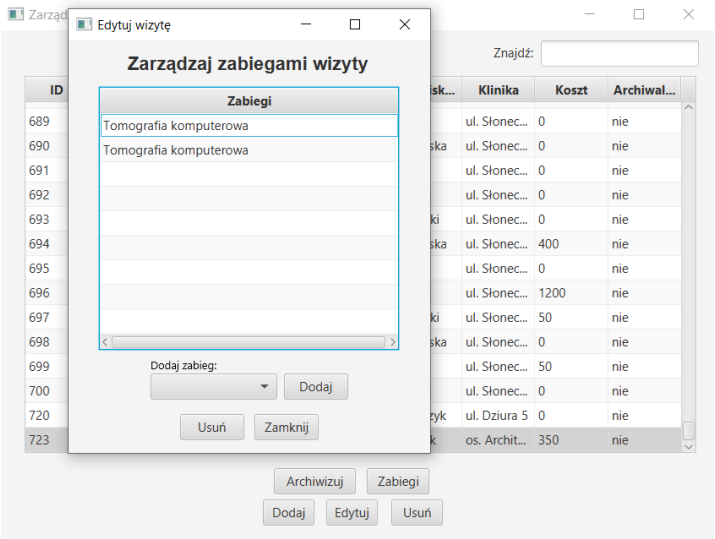
Przy próbie usunięcia nowododanej wizyty wyświetli się komunikat:



By móc usunąć nowy zabieg, należy najpierw usunąć automatycznie dodany zabieg-placeholder „Do ustalenia”. Jest to zabezpieczenie przed niechcianym usunięciem już dodanej wizyty. Zatem przechodzę do przycisku zabiegi:



Mogę tu dodać zabiegi które zostaną przeprowadzone podczas wizyty. Po dodaniu nowego zabiegu, placeholder „Do ustalenia” zniknie. Po zamknięciu okna zostanie zaktualizowany koszt całkowity zabiegu (procedura SUMAKOSZT). Po dodaniu dwóch zabiegów z tomografią komputerową pacjentka zapłaci za wizytę 350 zł, ponieważ tylko jedna tomografia jest refundowana w ciągu roku, a za drugą musi już zapłacić pełną kwotę (procedura REFUNDACJAPRZYSLUGUJEDOR).



Możliwe jest wyszukanie rekordów które zawierają dowolną rubrykę o podanej zawartości:

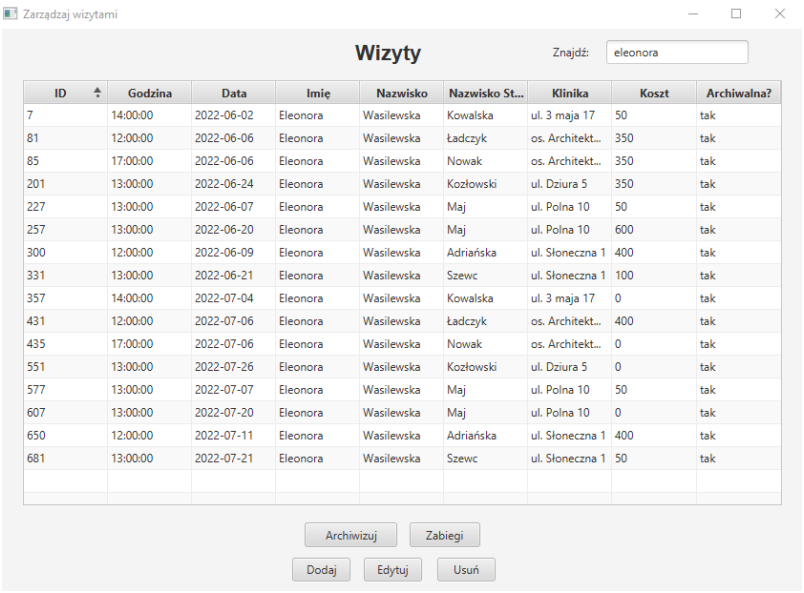
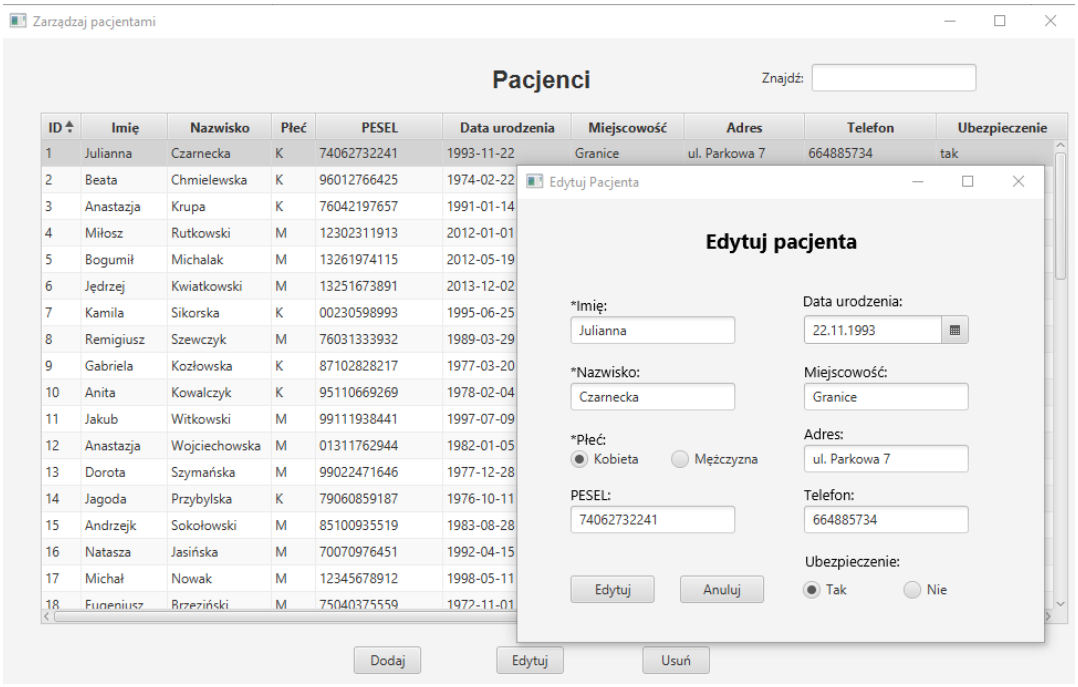


Tabela pacjenci również posiada przeszukiwanie kolumn, automatyczne wypełnienie pól edycji zaznaczonego pacjenta a także walidację danych wprowadzonych przez użytkownika.



Usunięcie pacjenta, który ma przypisane do siebie wizyty:

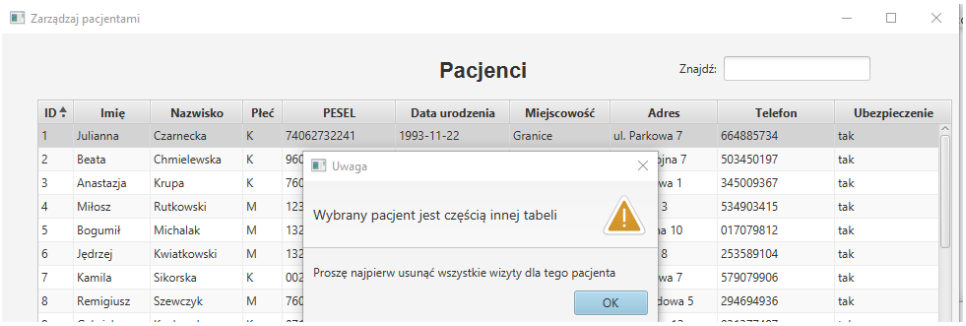
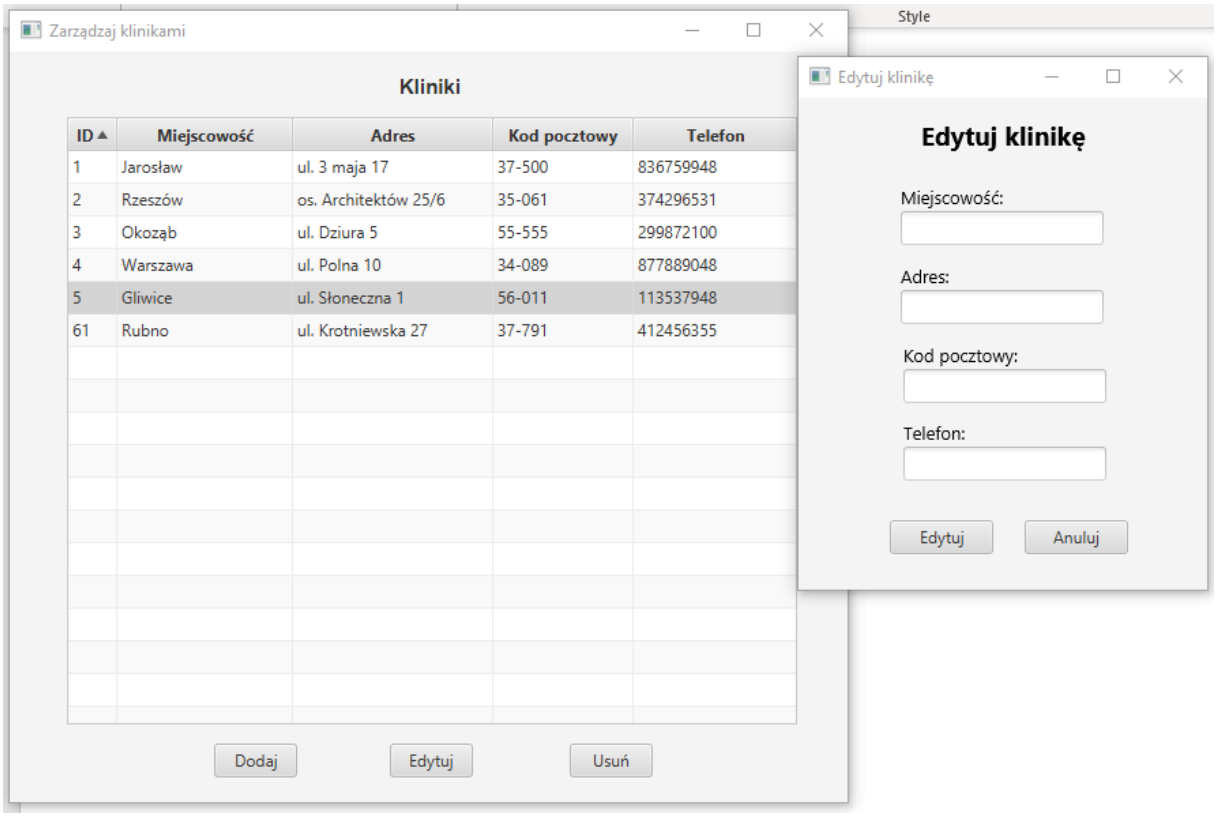
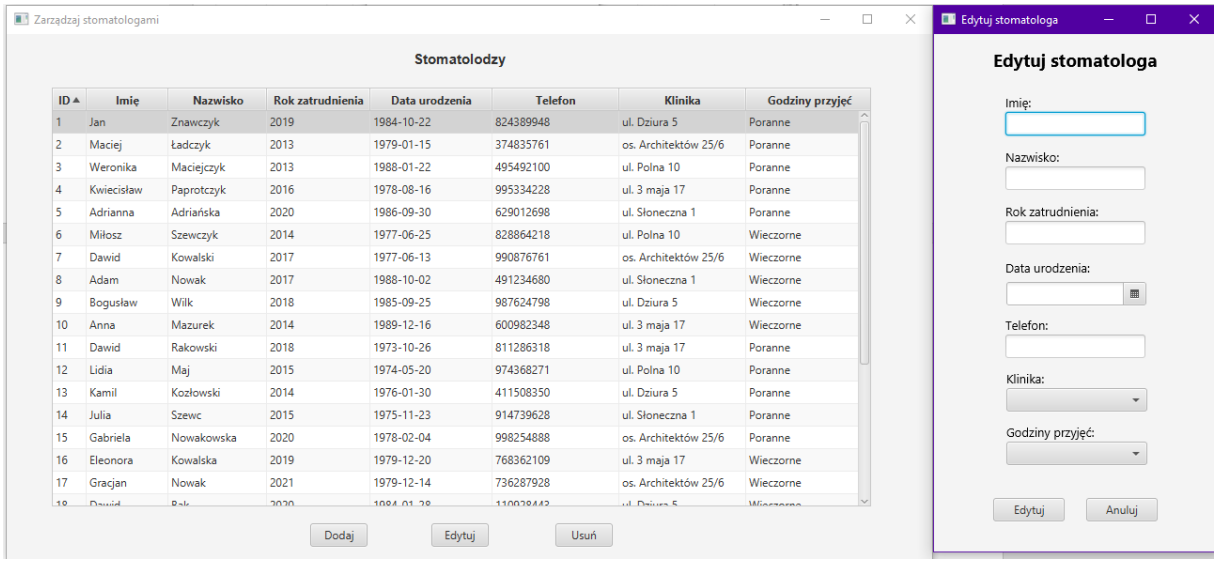


Tabela kliniki:



Dane edycji muszą zostać wprowadzone ręcznie. Zachodzi podstawowa walidacja (REGEX).

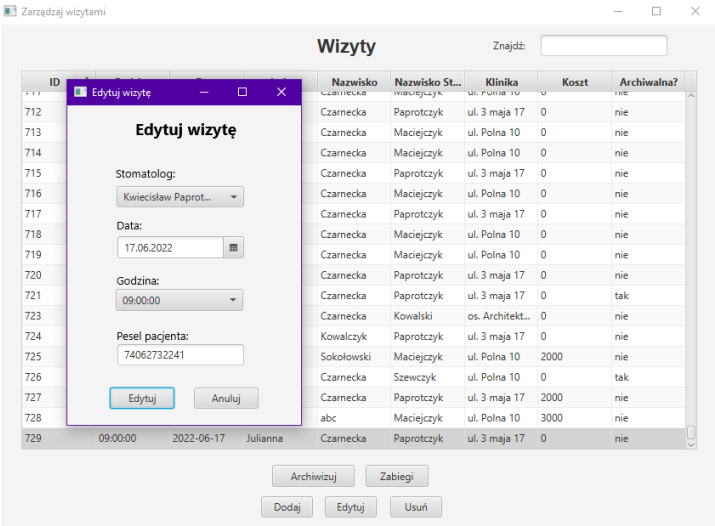
Tabela stomatologów:



Dane edycji muszą zostać wprowadzone ręcznie. Zachodzi podstawowa walidacja (REGEX).

Krótki opis kodu źródłowego:

Przedstawię przykład edycji wizyty:



- Zapełnienie pól okna edycji zaznaczonej wizyty

```
// pobierz dane zaznaczonej wizyty i uzupelnij nimi pole
String sql = "SELECT wizyta.id_wizyty,wizyta.godzina,wizyta.data_, pacjent.id_pacjenta AS idPac,"
        + " stomatolog.imie AS imStom, stomatolog.nazwisko AS nazStom"
        + " FROM wizyta "
        + " INNER JOIN pacjent ON wizyta.id_pacjenta=pacjent.id_pacjenta "
        + " INNER JOIN stomatolog ON wizyta.id_stomatologa=stomatolog.id_stomatologa "
        + " WHERE id_wizyty=?";

ResultSet result = handler.zaznaczonaWizyta(sql, wizytaSelectedId);
result.next();
idStomatologaTekst.setValue(String.valueOf(result.getString("imStom")) + " " + String.valueOf(result.getString(
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
LocalDate localDate = LocalDate.parse(String.valueOf(result.getDate("data_")), formatter);
DataPicker.setValue(localDate);

godzinaCombo.setItems(FXCollections.observableList(handler.listaDostepnychGodzin(DataPicker.getValue().toStrinc
godzinaCombo.setValue(result.getString("godzina"));
// znajdz pesel pacjenta klinika.id_pacjenta
int przekazId = result.getInt("idPac");
sql = "SELECT pesel FROM pacjent WHERE id_pacjenta = ?";
result = handler.znajdzPeselWizyta(sql, przekazId);
result.next();
peselPacjentaTekst.setText(result.getString(1));
```

Do obiektu DatabaseHandler handler (klasa zajmująca się łączeniem z bazą danych i finalnym wykonywaniem query) przekazuję widoczne na samej górze zapytanie sql. Zwróci mi ono zaznaczony rekord wizyty. To zapytanie zostaje przekazane do funkcji DatabaseHandler.zaznaczonaWizyta().

```
ResultSet zaznaczonaWizyta(String sql, int id) throws SQLException {
    createConnection(sql);
    stmt.setInt(1, id);
    ResultSet result = stmt.executeQuery();
    return result;
}
```

Funkcja DatabaseHandler.createConnection() pozwala połączyć się z bazą danych i zadeklarować procedurę (CallableStatement) lub zwykłe query (PreparedStatement).

```
//Metoda tworząca połączenie z bazą
void createConnection(String sql) {
    try {
        String url = "jdbc:oracle:thin:@localhost:1521:KOSMOS";
        String user = "jakub";
        String password = "melokawka";
        Class.forName("oracle.jdbc.OracleDriver").newInstance();
        Connection con = DriverManager.getConnection(url, user, password);
        if (sql.contains("call")) {
            call = con.prepareCall(sql);
        } else {
            stmt = con.prepareStatement(sql);
        }
```

Jeżeli w łańcuchu sql zostanie wykryte słowo kluczowe call, to program wie że to procedura.

Otrzymanymi wartościami wypełniam wszystkie pola poza peselem pacjenta.

By uzyskać pesel pacjenta wykonuję query z dolnej części pierwszego bloku kodu.

```
ResultSet znajdzPeselWizyta(String sql, int id) throws SQLException {
    createConnection(sql);
    stmt.setInt(1, id);
    ResultSet result = stmt.executeQuery();
    return result;
}
```

Program wreszcie przechodzi do edycji wizyty.

```
if (godzinaCombo.getValue() == null || peselPacjentaTekst.getText().equals("")
        || idStomatologaTekst.getValue().equals("") || DataPicker.getValue() == null) {
    Alert alert = new Alert(Alert.AlertType.WARNING);
    alert.setTitle("Uwaga");
    alert.setHeaderText("Nie wypełniono wymaganych pól");
    alert.setContentText("Proszę wypełnić wszystkie pola");
    alert.showAndWait();
} else {
    try {
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat("HH:mm:ss");
        long msec = simpleDateFormat.parse(godzinaCombo.getValue()).getTime();
        Time godz = new Time(msec);
        handler = DatabaseHandler.getInstance();
        sql = "{call wizyta_pack.edytuj(?,?,?,?,?,?,?)}";
        handler.edytujWizyte(sql, godz, DataPicker.getValue().toString(), peselPacjentaTekst.getText(),
                idStomatologaTekst.getValue().toString(), wizytaSelectedId);
        Stage stage = (Stage) DataPicker.getScene().getWindow();
        stage.close();
```

Jeżeli pola nie są puste, do handlera zostaje przekazane query wywołujące procedurę wizyta_pack.edytuj(). Do funkcji edytującej zostały przekazane: query, id zaznaczonej wizyty i wszystkie pola.

Wewnątrz handlera najpierw sprawdzam id stomatologa dla zaznaczonych w comboboxie imienia i nazwiska:

```
void edytujWizyte(String sql, Time godzina, String data, String pesel, Str
    String sql1;
    ResultSet result;
    String[] dane = stom.split(" ");
    sql1 = "SELECT id_stomatologa FROM stomatolog "
            + "WHERE stomatolog.imie=? AND stomatolog.nazwisko=?";
    createConnection(sql1);
    stmt.setString(1, dane[0]);
    stmt.setString(2, dane[1]);
    result = stmt.executeQuery();
    result.next();
    int id_stomatologa = result.getInt(1);
```

Następnie znajduję id_pacjenta dla wpisanego peselu.

```
String sql2;
sql2 = "SELECT id_pacjenta FROM pacjent WHERE pesel=?";
createConnection(sql2);
stmt.setString(1, pesel);
result = stmt.executeQuery();
// jezeli resultset zwroci false to nie istnieje taki pesel
int id_pacjenta;
try {
    result.next();
    id_pacjenta = result.getInt(1);
} catch (Exception e) {
    throw new IllegalArgumentException();
}
```

Konwertuję zaznaczoną w datepickerze (lub wpisaną ręcznie) datę na akceptowalną przez bazę danych.

```
createConnection(sql);
call.setInt(1, id_wizyty);
call.setString(2, godzina.toString());
CharSequence seq = data;
call.setString(3, LocalDate.parse(
        seq,
        DateTimeFormatter.ofPattern("uuuu-MM-dd")
).format(
        DateTimeFormatter.ofPattern("dd-MM-uuuu")
));
call.setInt(4, id_pacjenta);
call.setInt(5, id_stomatologa);
```

Pobieram koszt wizyty.

```
sql1 = "SELECT koszt FROM wizyta WHERE id_wizyty=?";
createConnection(sql1);
stmt.setInt(1, id_wizyty);
result = stmt.executeQuery();
result.next();
int koszt = result.getInt(1);

call.setInt(6, koszt);
```

Pobieram wartość „archiwalna” i wykonuję procedurę edycji.

```
sql1 = "SELECT archiwalna FROM wizyta WHERE id_wizyty=?";
createConnection(sql1);
stmt.setInt(1, id_wizyty);
result = stmt.executeQuery();
result.next();
String archiwalna = result.getString(1);

call.setString(7, archiwalna);
call.execute();
```