Baza danych - Piłkarze
Dokumentacja

Paweł Warcaba

II rok Informatyka

Grupa 4

1. Dokładny opis bazy danych

- Baza danych została utworzono z sześciu encji (tabel) - takich jak:

* Pilkarz – informacje o piłkarzu (jest główną tabelą na której opiera się baza danych),

* Kartki – liczba otrzymanych kartek dla konkretnego piłkarza,

* Reprezentacja – reprezentacja dla piłkarza,

* Klub – kluby dla piłkarza,

* Stadion – stadiony dla konkretnego klubu,

* Trener – trenerzy dla konkretnego klubu

- Relacje:

* Tabela Pilkarz:

    * jest w relacji 1:1 z tabelą Kartki,

    * jest w relacji ∞:1 z tabelą Reprezentacja,

    * jest w relacji ∞:1 z tabelą Klub

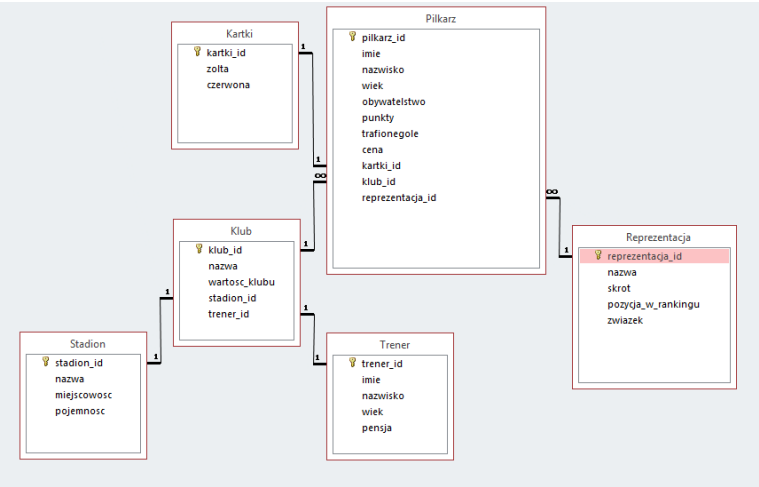
* Tabela Klub:

    * jest w relacji 1:1 z tabelą Stadion,

    * jest w relacji 1:1 z tabelą Trener,

Dla więzów integralności przyjęto strategię ON DELETE: RESTRICT – oznacza ona że przy próbie usunięcia rekordu wyświetlany jest odpowiedni komunikat o braku możliwości wykonania tej czynności. Dzięki interfejsowi możemy dodawać, modyfikować, usuwać i wyświetlać dane w tabelach. Baza danych skupia się głównie na obliczeniach. Przykładowymi obliczeniami mogą być punkty i cena (biorąc pod uwagę „karne” kartki) - dla piłkarza, pensja – dla trenera, pozycja w rankingu - dla reprezentacji czy wartość - dla klubu. Warto wspomnieć że dostęp do bazy danych działa jeśli podamy poprawną nazwę użytkownika i hasło. W moim projekcie użytkownik posiada nazwę: **pawel**, zaś hasło: **war**.

2. Schemat ERD bazy danych z relacjami

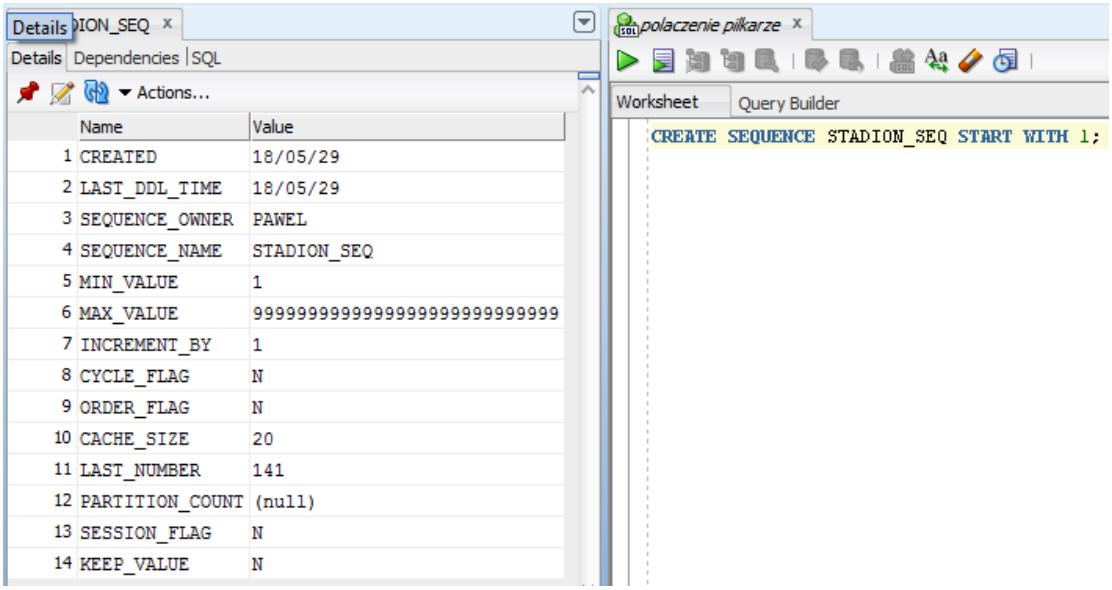


3. Funkcjonalności PL/SQL

Funkcjonalności jakie zawarłem w projekcie opierają się na zapytaniach SQL i bardziej złożonych zapytaniach PL/SQL które przedstawię poniżej:

A) Sekwencje:

Oracle potrzebuje do autoinkrementacji implementacje sekwencji, przykład przedstawiłem poniżej. Sekwencja została wykonana dla każdej z sześciu tabel.



Dzięki temu zabiegowi za każdym razem gdy dodamy nowy rekord do konkretnej tabeli jego id jest unikalne.

B) Triggery:

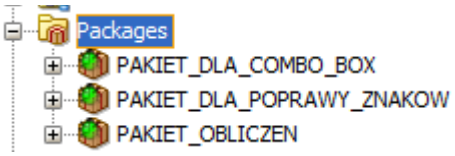
W moim projekcie zaimplementowałem również Triggery – są to wyzwalacze „aktywowane” podczas jakiegoś zdarzenia. Poniżej przedstawiłem przykładowy Trigger, który zastosowałem do każdej z sześciu tabel.

```
create or replace
trigger PILKARZ_TRI
BEFORE INSERT ON PILKARZ
FOR EACH ROW
BEGIN
  SELECT PILKARZ_SEQ.nextval
  INTO :new.PILKARZ_ID
  FROM dual;

END;
```

Dzięki temu zabiegowi za każdym razem gdy dodamy rekord do tabeli (poleceniem INSERT) zostanie nadana kolejna wartość id z utworzonych wcześniej sekwencji.

C) Pakiety:

Aby rozdzielić funkcjonalności dla prostszego wyszukiwania, a także dla logicznego podzielenia funkcjonalności stworzyłem pakiety. W trzech pakietach zawarłem odpowiednie Funkcje i Procedury wykonujące określone zadania. Podział wygląda następująco:



*** PAKIET_DLA_COMBO_BOX**

W tym pakiecie zawarłem trzy funkcje, których dokładny opis zawarłem na poniższym screenie:

```
create or replace
package body PAKIET_DLA_COMBO_BOX
is
    function COMBO_BOX_REPREZENTACJA return SYS_REFCURSOR as
     KURSOR SYS_REFCURSOR; -- funkcja zwracająca kursor
    begin
     OPEN KURSOR FOR SELECT nazwa FROM reprezentacja; -- otwarcie kursora dla pola nazwa z tabeli reprezentacja
        return KURSOR;
    end COMBO_BOX_REPREZENTACJA;

    function COMBO_BOX_KLUB return SYS_REFCURSOR as
     KURSOR SYS_REFCURSOR; -- funkcja zwracająca kursor
    begin
     OPEN KURSOR FOR SELECT nazwa FROM klub; -- otwarcie kursora dla pola nazwa z tabeli klub
        return KURSOR;
    end COMBO_BOX_KLUB;

    function COMBO_BOX_PILKARZ return SYS_REFCURSOR as
     KURSOR SYS_REFCURSOR; -- funkcja zwracająca kursor
    begin
     OPEN KURSOR FOR SELECT imie FROM pilkarz; -- otwarcie kursora dla pola imie z tabeli pilkarz
        return KURSOR;
    end COMBO_BOX_PILKARZ;

end;
```

Podsumowując pakiet można stwierdzić że jego głównym zadaniem jest pobranie odpowiedniej kolumny do kursora i zwrócenie jej jako kursor do interfejsu, a co za tym idzie dzięki temu zabiegowi aktualizacja combo_boxów. Warto wspomnieć że funkcje w Oracle muszą coś zwracać.

*** PAKIET_DLA_POPRAWY_ZNAKOW**

W kolejnym pakiecie zawarłem sześć Procedur, którymi głównym zadaniem jest poprawienie tekstu wprowadzonego przez użytkownika na pierwszą duża literę (np. Imię) lub w innym przypadku zamiana na wszystkie duże litery (np. skrót).

- AKTUALIZACJA_REPREZENTACJI – poprawa wszystkich znaków dla tabeli reprezentacja

```
procedure AKTUALIZACJA_REPREZENTACJI is
DANY_WIERSZ REPREZENTACJA%rowtype; -- zmienna typu zlozonego rowtype
zmienna_skrot varchar2(255); -- zmienna varchar2
cursor KURSOR is
select * from reprezentacja; -- pobranie do kursora wszystkich kolumn z tabeli reprezentacja
begin
    for DANY_WIERSZ in KURSOR loop   -- zmiana w petli odpowiednich zawartosci w kolumnach i ich aktualizacja
      zmienna_skrot := DANY_WIERSZ.skrot;

      UPDATE reprezentacja SET skrot=upper(zmienna_skrot) WHERE reprezentacja_id=DANY_WIERSZ.reprezentacja_id;
      DBMS_OUTPUT.put_line(zmienna_skrot);
    end loop;
end;
```

- AKTUALIZACJA_REPREZENTACJI_2 – poprawa pierwszego znaku dla tabeli reprezentacja

```
 procedure AKTUALIZACJA_REPREZENTACJI_2 is
DANY_WIERSZ REPREZENTACJA%rowtype; -- zmienna typu zlozonego rowtype
zmienna_znak varchar2(255);  -- zmienna varchar2
zmienna_znak2 varchar2(255);  -- zmienna varchar2

cursor KURSOR is
select * from reprezentacja; -- pobranie do kursora wszystkich kolumn z tabeli reprezentacja
begin
    for DANY_WIERSZ in KURSOR loop    -- zmiana w petli odpowiednich zawartosci w kolumnach i ich aktualizacja
      zmienna_znak := DANY_WIERSZ.nazwa;
      zmienna_znak2 := DANY_WIERSZ.zwiazek;

      UPDATE reprezentacja SET
      nazwa=upper(SUBSTR(zmienna_znak,1,1)) || LOWER(SUBSTR(zmienna_znak,2)),
      zwiazek=upper(SUBSTR(zmienna_znak2,1,1)) || LOWER(SUBSTR(zmienna_znak2,2))
      WHERE reprezentacja_id=DANY_WIERSZ.reprezentacja_id;
      DBMS_OUTPUT.put_line(zmienna_znak);
    end loop;
end;
```

- AKTUALIZACJA_PILKARZ – poprawa pierwszego znaku dla tabeli pilkarz

```
 procedure AKTUALIZACJA_PILKARZ is
DANY_WIERSZ PILKARZ%rowtype;  -- zmienna typu zlozonego rowtype
zmienna_znak varchar2(255);  -- zmienna varchar2
zmienna_znak2 varchar2(255);  -- zmienna varchar2
zmienna_znak3 varchar2(255);  -- zmienna varchar2
cursor KURSOR is
select * from pilkarz; -- pobranie do kursora wszystkich kolumn z tabeli pilkarz
begin
    for DANY_WIERSZ in KURSOR loop    -- zmiana w petli odpowiednich zawartosci w kolumnach i ich aktualizacja
      zmienna_znak := DANY_WIERSZ.imie;
      zmienna_znak2 := DANY_WIERSZ.nazwisko;
      zmienna_znak3 := DANY_WIERSZ.obywatelstwo;

      UPDATE pilkarz SET
      imie=upper(SUBSTR(zmienna_znak,1,1)) || LOWER(SUBSTR(zmienna_znak,2)),
      nazwisko=upper(SUBSTR(zmienna_znak2,1,1)) || LOWER(SUBSTR(zmienna_znak2,2)),
      obywatelstwo=upper(SUBSTR(zmienna_znak3,1,1)) || LOWER(SUBSTR(zmienna_znak3,2))
      WHERE pilkarz_id=DANY_WIERSZ.pilkarz_id;
      DBMS_OUTPUT.put_line(zmienna_znak);
    end loop;
end;
```

- AKTUALIZACJA_KLUB – poprawa pierwszego znaku dla tabeli klub

```
procedure AKTUALIZACJA_KLUB is
DANY_WIERSZ KLUB%rowtype; -- zmienna typu zlozonego rowtype
zmienna_znak varchar2(255);  -- zmienna varchar2

cursor KURSOR is
select * from klub; -- pobranie do kursora wszystkich kolumn z tabeli klub
begin
    for DANY_WIERSZ in KURSOR loop    -- zmiana w petli odpowiednich zawartosci w kolumnach i ich aktualizacja
      zmienna_znak := DANY_WIERSZ.nazwa;

      UPDATE klub SET
      nazwa=upper(SUBSTR(zmienna_znak,1,1)) || LOWER(SUBSTR(zmienna_znak,2))
      WHERE klub_id=DANY_WIERSZ.klub_id;
      DBMS_OUTPUT.put_line(zmienna_znak);
    end loop;
end;
```

- AKTUALIZACJA_TRENER – poprawa pierwszego znaku dla tabeli trener

```
 procedure AKTUALIZACJA_TRENER is
DANY_WIERSZ TRENER%rowtype; -- zmienna typu zlozonego rowtype
zmienna_znak varchar2(255);  -- zmienna varchar2
zmienna_znak2 varchar2(255);  -- zmienna varchar2
cursor KURSOR is
select * from trener; -- pobranie do kursora wszystkich kolumn z tabeli trener
begin
    for DANY_WIERSZ in KURSOR loop    -- zmiana w petli odpowiednich zawartosci w kolumnach i ich aktualizacja
      zmienna_znak := DANY_WIERSZ.imie;
      zmienna_znak2 := DANY_WIERSZ.nazwisko;

      UPDATE trener SET
      imie=upper(SUBSTR(zmienna_znak,1,1)) || LOWER(SUBSTR(zmienna_znak,2)),
      nazwisko=upper(SUBSTR(zmienna_znak2,1,1)) || LOWER(SUBSTR(zmienna_znak2,2))
      WHERE trener_id=DANY_WIERSZ.trener_id;
      DBMS_OUTPUT.put_line(zmienna_znak);
    end loop;
end;
```

- AKTUALIZACJA_STADION – poprawa pierwszego znaku dla tabeli stadion

```
 procedure AKTUALIZACJA_STADION is
DANY_WIERSZ STADION%rowtype; -- zmienna typu zlozonego rowtype
zmienna_znak varchar2(255);  -- zmienna varchar2
zmienna_znak2 varchar2(255);  -- zmienna varchar2
cursor KURSOR is
select * from stadion; -- pobranie do kursora wszystkich kolumn z tabeli stadion
begin
    for DANY_WIERSZ in KURSOR loop    -- zmiana w petli odpowiednich zawartosci w kolumnach i ich aktualizacja
      zmienna_znak := DANY_WIERSZ.nazwa;
      zmienna_znak2 := DANY_WIERSZ.miejscowosc;

      UPDATE stadion SET
      nazwa=upper(SUBSTR(zmienna_znak,1,1)) || LOWER(SUBSTR(zmienna_znak,2)),
      miejscowosc=upper(SUBSTR(zmienna_znak2,1,1)) || LOWER(SUBSTR(zmienna_znak2,2))
      WHERE stadion_id=DANY_WIERSZ.stadion_id;
      DBMS_OUTPUT.put_line(zmienna_znak);
    end loop;
end;
```

Podsumowując pakiet można stwierdzić że jego głównym zadaniem jest odpowiednia „poprawa” wpisanych danych do bazy danych. Podczas wprowadzania danych, dane te są „poprawiane” przy

użyciu odpowiednich słów kluczowych w Oraclu np. UPPER. Warto wspomnieć że procedury w Oracle nic nie zwracają.

*** PAKIET_OBLICZEN**

W ostatnim pakiecie zawarłem również sześć procedur, którymi głównym zadaniem są dogłębne obliczenia na konkretnych danych.

- OBLICZENIA_PILKARZ_UPDATE – obliczenie punktów

```
procedure OBLICZENIA_PILKARZ_UPDATE is
    DANY_WIERSZ PILKARZ%rowtype; -- zmienna typu zlozonego rowtype
    zmienna_punkty number(38,0); -- zmienna typu number
    cursor KURSOR is
    select * from pilkarz; -- pobranie do kursora wszystkich kolumn z tabeli pilkarz
    begin
        for DANY_WIERSZ in KURSOR loop -- wykonanie obliczen w petli dla kazdego rekordu w tabeli pilkarz (obliczenie punktow - trafionygol*10 )
          zmienna_punkty := DANY_WIERSZ.trafionegole*10;
          UPDATE pilkarz SET punkty=zmienna_punkty WHERE pilkarz_id=DANY_WIERSZ.pilkarz_id;
          DBMS_OUTPUT.put_line(zmienna_punkty);
        end loop;
    end;
```

- OBLICZENIA_PILKARZ_UPDATE_2 – obliczenie ceny

```
procedure OBLICZENIA_PILKARZ_UPDATE_2 is
    DANY_WIERSZ PILKARZ%rowtype;  -- zmienna typu zlozonego rowtype
    zmienna_cena decimal(38,0); -- zmienna typu decimal
    zmienna_ile decimal(38,0); -- zmienna typu decimal
    cursor KURSOR is
    select * from pilkarz;  -- pobranie do kursora wszystkich kolumn z tabeli pilkarz
    begin
        for DANY_WIERSZ in KURSOR loop -- wykonanie obliczen w petli dla kazdego rekordu w tabeli pilkarz (obliczenie ceny - za kazde 100 pkt pilkarz otrzymuje 500 zl)
        zmienna_ile:= FLOOR(DANY_WIERSZ.punkty/100);
        IF zmienna_ile<1
          THEN zmienna_cena:= 0;
          ELSE zmienna_cena:= zmienna_ile*500;
        END IF;
          UPDATE pilkarz SET cena=zmienna_cena WHERE pilkarz_id=DANY_WIERSZ.pilkarz_id;
          DBMS_OUTPUT.put_line(zmienna_cena);
        end loop;
    end;
```

- OBLICZENIA_KARTEK_UPDATE – obliczenie ceny po wzięciu pod uwagę „karnych” kartek

```
 procedure OBLICZENIA_KARTEK_UPDATE is
    DANY_WIERSZ PILKARZ%rowtype;  -- zmienna typu zlozonego rowtype
    zmienna_cena number(38,0);  -- zmienna typu number
    zmienna_cenal number(38,0);  -- zmienna typu number
    zmienna_cena2 number(38,0);  -- zmienna typu number

    cursor KURSOR is
    select * from pilkarz;  -- pobranie do kursora wszystkich kolumn z tabeli pilkarz

    DANY_WIERSZ2 KARTKI%rowtype;  -- kolejna zmienna typu zlozonego rowtype
    TYPE typ_wynik_cena_temp IS TABLE OF kartki.zolta%TYPE INDEX BY BINARY_INTEGER; -- typ tablicowy do przechowywania liczby zoltych kartek
    TYPE typ_wynik_cena_temp2 IS TABLE OF kartki.czerwona%TYPE INDEX BY BINARY_INTEGER;  -- typ tablicowy do przechowywania liczby czerwonych kartek

    wynik_cena_temp typ_wynik_cena_temp; --deklaracja zmiennej typu tablicowego
    wynik_cena_temp2 typ_wynik_cena_temp2; --deklaracja zmiennej typu tablicowego


    cursor KURSOR2 is
    select * from kartki;  -- pobranie do drugiego kursora wszystkich kolumn z tabeli kartki

    begin

        for DANY_WIERSZ2 in KURSOR2 loop  -- przypisanie danych w petli do typu tablicowego
        wynik_cena_temp(DANY_WIERSZ2.kartki_id):=DANY_WIERSZ2.zolta;
        wynik_cena_temp2(DANY_WIERSZ2.kartki_id):=DANY_WIERSZ2.czerwona;
        --DBMS_OUTPUT.put_line(wynik_cena_temp(DANY_WIERSZ2.kartki_id));
```

```
    end loop;

    for DANY_WIERSZ in KURSOR loop -- wykonanie obliczen w petli dla kazdego rekordu w tabeli pilkarz
    zmienna_cena:= DANY_WIERSZ.cena;

        IF wynik_cena_temp(DANY_WIERSZ.kartki_id)>0 -- najpierw przeprwodzamy obliczenia dla kartki czerwonej
        THEN zmienna_cena1:=DANY_WIERSZ.cena*(0.25*wynik_cena_temp(DANY_WIERSZ.kartki_id)); -- czerwona - za kazda kartke 25% ceny traci dany pilkarz
        ELSE zmienna_cena1:=0; -- nie odejmuje dla 0
        END IF;
         DBMS_OUTPUT.put_line('1:'||zmienna_cena1);
        IF wynik_cena_temp2(DANY_WIERSZ.kartki_id)>0 -- nastepnie przeprwodzamy obliczenia dla kartki zoltej
        THEN zmienna_cena2:=DANY_WIERSZ.cena*(0.05*wynik_cena_temp2(DANY_WIERSZ.kartki_id)); -- zolta - za kazda kartke 5% ceny traci dany pilkarz
        ELSE zmienna_cena2:=0; -- nie odejmuje dla 0
        END IF;
         DBMS_OUTPUT.put_line('2:'||zmienna_cena2);

        zmienna_cena:=zmienna_cena-(zmienna_cena1+zmienna_cena2);   -- sumujemy odejmowana wartosc przez "karne" kartki, a następnie odejmujemy od aktualnej ceny

        IF zmienna_cena<0  -- zabezpieczenie przed ujemna cena
        THEN zmienna_cena:=0;
        END IF;

    UPDATE pilkarz SET cena=zmienna_cena WHERE pilkarz_id=DANY_WIERSZ.pilkarz_id;  -- aktualizacja ceny konkretnego pilkarza
    DBMS_OUTPUT.put_line('wynik:'||zmienna_cena);
    end loop;
  end;
```

- OBLICZENIA_REPREZENT_UPDATE – obliczenie pozycji za każdy gol wybranego piłkarza

```
procedure OBLICZENIA_REPREZENT_UPDATE is
DANY_WIERSZ PILKARZ%rowtype; -- zmienna typu zlozonego rowtype
zmienna_punkty number(38,0); -- zmienna typu number
czyinny number(38,0); -- zmienna typu number
cursor KURSOR is
select * from pilkarz order by reprezentacja_id ASC;  -- pobranie do kursora wszystkich (posortowanych po id) kolumn z tabeli pilkarz
begin
    zmienna_punkty:=0;
    czyinny:=DANY_WIERSZ.reprezentacja_id;
    for DANY_WIERSZ in KURSOR loop -- wykonanie obliczen w petli dla kazdego rekordu w tabeli pilkarz


      IF czyinny = DANY_WIERSZ.reprezentacja_id -- sprawdzam czy zliczam dane innego pilkarza
      THEN
      zmienna_punkty := zmienna_punkty+DANY_WIERSZ.trafionegole*25; -- za kazdy gol (kazdego wybranego przez nia pilkarza) reprezentacja dostaje 25pkt
      UPDATE reprezentacja SET pozycja_w_rankingu=zmienna_punkty WHERE reprezentacja_id=DANY_WIERSZ.reprezentacja_id;
      DBMS_OUTPUT.put_line(zmienna_punkty);
      ELSE

      zmienna_punkty:=0;
      czyinny:=DANY_WIERSZ.reprezentacja_id;
      zmienna_punkty := zmienna_punkty+DANY_WIERSZ.trafionegole*25; -- za kazdy gol (kazdego wybranego przez nia pilkarza) reprezentacja dostaje 25pkt
      UPDATE reprezentacja SET pozycja_w_rankingu=zmienna_punkty WHERE reprezentacja_id=DANY_WIERSZ.reprezentacja_id;
      DBMS_OUTPUT.put_line(zmienna_punkty);
      END IF;
    end loop;
end;
```

- OBLICZENIA_KLUB_UPDATE – obliczenie wartości klubu

```
procedure OBLICZENIA_KLUB_UPDATE is

DANY_WIERSZ PILKARZ%rowtype;  -- zmienna typu zlozonego rowtype
zmienna_ceny number(19,4);  -- zmienna typu number
czyinny number(38,0);  -- zmienna typu number
cursor KURSOR is
select * from pilkarz order by klub_id ASC;  -- pobranie do kursora wszystkich (posortowanych po id) kolumn z tabeli pilkarz
begin
    zmienna_ceny:=0;
    czyinny:=DANY_WIERSZ.klub_id;
    for DANY_WIERSZ in KURSOR loop -- wykonanie obliczen w petli dla kazdego rekordu w tabeli pilkarz


        IF czyinny = DANY_WIERSZ.klub_id -- sprawdzam czy zliczam dane innego pilkarza
        THEN
        zmienna_ceny := zmienna_ceny+DANY_WIERSZ.cena*0.05; -- zliczam 5% ceny kazdego pilkarza, ktory jest w tym klubie
        UPDATE klub SET wartosc_klubu=zmienna_ceny WHERE klub_id=DANY_WIERSZ.klub_id;
        DBMS_OUTPUT.put_line(zmienna_ceny);
        ELSE

        zmienna_ceny:=0;
        czyinny:=DANY_WIERSZ.klub_id;
        zmienna_ceny := zmienna_ceny+DANY_WIERSZ.cena*0.05; -- zliczam 5% ceny kazdego pilkarza, ktory jest w tym klubie
        UPDATE klub SET wartosc_klubu=zmienna_ceny WHERE klub_id=DANY_WIERSZ.klub_id;
        DBMS_OUTPUT.put_line(zmienna_ceny);
        END IF;
    end loop;

end;
```

- OBLICZENIA_TRENER_UPDATE – obliczenie pensji trenera

```
procedure OBLICZENIA_TRENER_UPDATE is
DANY_WIERSZ KLUB%rowtype;  -- zmienna typu zlozonego rowtype
zmienna_ceny number(19,4); -- zmienna typu number
czyinny number(38,0); -- zmienna typu number
cursor KURSOR is
select * from klub order by trener_id ASC; -- pobranie do kursora wszystkich (posortowanych po id) kolumn z tabeli klub
begin
    zmienna_ceny:=0;
    czyinny:=DANY_WIERSZ.trener_id;
    for DANY_WIERSZ in KURSOR loop  -- wykonanie obliczen w petli dla kazdego rekordu w tabeli pilkarz

        IF czyinny = DANY_WIERSZ.trener_id  -- sprawdzam czy zliczam dane innego pilkarza
        THEN
        zmienna_ceny := zmienna_ceny+DANY_WIERSZ.wartosc_klubu*2; -- podwojona wartosc klubu tworzy pensje, jednak nie moze przekroczyc 10tys
         IF zmienna_ceny>10000
         THEN zmienna_ceny:=10000;
         END IF;
        UPDATE trener SET pensja=zmienna_ceny WHERE trener_id=DANY_WIERSZ.trener_id;
        DBMS_OUTPUT.put_line(zmienna_ceny);
        ELSE

        zmienna_ceny:=0;
        czyinny:=DANY_WIERSZ.trener_id;
          zmienna_ceny := zmienna_ceny+DANY_WIERSZ.wartosc_klubu*2;  -- podwojona wartosc klubu tworzy pensje, jednak nie moze przekroczyc 10tys
        IF zmienna_ceny>10000
         THEN zmienna_ceny:=10000;
         END IF;
        UPDATE trener SET pensja=zmienna_ceny WHERE trener_id=DANY_WIERSZ.trener_id;
        DBMS_OUTPUT.put_line(zmienna_ceny);
        END IF;
    end loop;
end;
```

Podsumowując ostatni pakiet można stwierdzić że jego głównym zadaniem jest prowadzenie obliczeń na dostarczonych mu danych. Podczas wprowadzania danych, dane te są odpowiednio liczone jak zostało to opisane przy każdej procedurze. W przedstawionych procedurach zastosowałem typy złożone (tablicowe, rekordowe).

4. Opis interfejsu

Aplikacje stworzyłem przy użyciu języka Java w środowisku Netbeans. Interfejs składa się z zakładek takich jak: dodawanie, zmodyfikowanie i usuwanie razem z wyświetlaniem w tabelach

Na początku warto wspomnieć jak wygląda wywołanie niektórych procedur:

```
    CallableStatement cst;

cst = polaczenie.prepareCall("{call PAKIET_OBLICZEN.OBLICZENIA_PILKARZ_UPDATE}");
 cst.execute();
 cst = polaczenie.prepareCall("{call PAKIET_OBLICZEN.OBLICZENIA_PILKARZ_UPDATE_2}");
 cst.execute();
 cst = polaczenie.prepareCall("{call PAKIET_OBLICZEN.OBLICZENIA_KARTEK_UPDATE}");
 cst.execute();
 cst = polaczenie.prepareCall("{call PAKIET_OBLICZEN.OBLICZENIA_REPREZENT_UPDATE}");
 cst.execute();
cst = polaczenie.prepareCall("{call PAKIET_OBLICZEN.OBLICZENIA_KLUB_UPDATE}");
cst.execute();

 cst = polaczenie.prepareCall("{call PAKIET_OBLICZEN.OBLICZENIA_TRENER_UPDATE}");
 cst.execute();
```

To jest przykładowe wywołanie procedur z pakietu PAKIET_OBLICZEN w ten sposób są przeprowadzone obliczenia na danych

```
cst = polaczenie.prepareCall("{?=call PAKIET_DLA_COMBO_BOX.COMBO_BOX_REPREZENTACJA()}");
cst.registerOutParameter(1, OracleTypes.CURSOR);
cst.execute();

ResultSet result = (ResultSet)cst.getObject(1);
while(result.next())
{
    //System.out.println(result.getString(1));
    jComboBox3.addItem(result.getString(1));
    jComboBox6.addItem(result.getString(1));
    jComboBox9.addItem(result.getString(1));
}
```

Tutaj przedstawiłem inny przykład, w tym przypadku widzimy wywołanie funkcji zwracającej kursor, a następnie wstawienie go do ComboBox w javie.

```
if(jTextField7.getText().equals("") || jTextField6.getText().equals("") || jTextField5.getText().equals(""))
{
    JOptionPane.showMessageDialog(null,"Nie podano wszystkich parametrów!");
}
else
{
        statement.executeUpdate("INSERT INTO REPREZENTACJA (nazwa, skrot, zwiazek) "
        + "VALUES ('"+jTextField7.getText()+"','"+jTextField6.getText()+"','"+jTextField5.getText()+"')");

        JOptionPane.showMessageDialog(null,"Poprawnie dodano rekord do bazy danych!");

        // dla skrotow w tabeli reprezentacja
         CallableStatement cst = polaczenie.prepareCall("{call PAKIET_DLA_POPRAWY_ZNAKOW.AKTUALIZACJA_REPREZENTACJI}");
         cst.execute();


           // poprawa znakow nazwa, zwiazek
           cst = polaczenie.prepareCall("{call PAKIET_DLA_POPRAWY_ZNAKOW.AKTUALIZACJA_REPREZENTACJI_2}");
           cst.execute();
```

Przykładowa opcja wczytania klubu do tabeli w interfejsie. Najpierw tworzymy połączenie a następnie zczytujemy dane umieszczając je w tabeli. Na koniec wywołujemy procedury PL/SQL.

```
// wczytaj klub
try
{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection polaczenie = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:pilkarze","pawel","war");  // polaczenie z baza danych
    Statement statement = polaczenie.createStatement();
            ((DefaultTableModel)jTable8.getModel()).setRowCount(0); // wyczyszczenie tabeli
            ResultSet wynik = statement.executeQuery("SELECT stadion_id, nazwa, miejscowosc, pojemnosc FROM STADION");    // wywolanie zapytania SQL
            while(wynik.next()) // pobranie kolejna wierszy
           {
            Object[] dane = new Object[4];
            dane[0]=wynik.getObject("stadion_id");
            dane[1]=wynik.getObject("nazwa");
            dane[2]=wynik.getObject("miejscowosc");
            dane[3]=wynik.getObject("pojemnosc");

           ((DefaultTableModel)jTable8.getModel()).addRow(dane); // wstawienie do tabeli

           }

            JOptionPane.showMessageDialog(null,"Poprawnie wczytano rekordy z bazy danych!");

    polaczenie.close();
}
catch(Exception e)
{
    JOptionPane.showMessageDialog(null,"Wystąpił wyjątek/błąd: "+e);
}
aktualizacja_procedur_PLSQL(); // wywolanie odpowiednich procedur PL/SQL
```

Przykładowa opcja dodania reprezentacji do tabeli w interfejsie. Najpierw tworzymy połączenie a następnie dodajemy dane (muszą być poprawne) do bazy danych. Na koniec wywołujemy odpowiednie procedury PL/SQL.

```
if(jTextField11.getText().equals("") || jTextField10.getText().equals("") || jTextField9.getText().equals("") || jComboBox3.getSelectedItem() == null)
{
    JOptionPane.showMessageDialog(null,"Nie podano wszystkich parametrów!");
}
else
{
    statement.executeUpdate
    ("UPDATE REPREZENTACJA SET nazwa='"+jTextField11.getText()+"', skrot='"+jTextField10.getText()+"', zwiazek='"+jTextField9.getText()+"' "
        + "WHERE nazwa='"+jComboBox3.getSelectedItem()+"'");

       // dla skrotow w tabeli reprezentacja
    CallableStatement cst = polaczenie.prepareCall("{call PAKIET_DLA_POPRAWY_ZNAKOW.AKTUALIZACJA_REPREZENTACJI}");
    cst.execute();


            // poprawa znakow nazwa, zwiazek
            cst = polaczenie.prepareCall("{call PAKIET_DLA_POPRAWY_ZNAKOW.AKTUALIZACJA_REPREZENTACJI_2}");
            cst.execute();


    JOptionPane.showMessageDialog(null,"Poprawnie zmodyfikowano reprezentacje w bazie danych!");
```

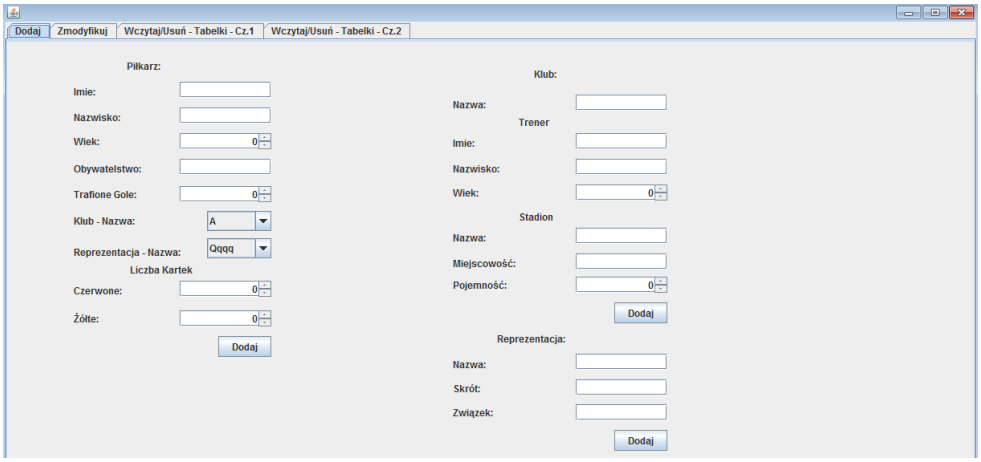
Przykładowa opcja zmodyfikowania reprezentacji w bazie danych. Najpierw tworzymy połączenie a następnie po poprawnie wprowadzonych danych następuje aktualizacja. Na koniec wywołujemy odpowiednie procedury PL/SQL.

```
        int[] rows = jTable6.getSelectedRows();

        for( int i = 0 ; i<rows.length; i++)
        {
        statement.executeUpdate
                ("DELETE FROM TRENER WHERE trener_id='"+jTable6.getModel().getValueAt(rows[i]-i,0)+"'");

        ((DefaultTableModel)jTable6.getModel()).removeRow(rows[i]-i);

        }

        JOptionPane.showMessageDialog(null,"Poprawnie usunięto trenera w bazie danych!");
```

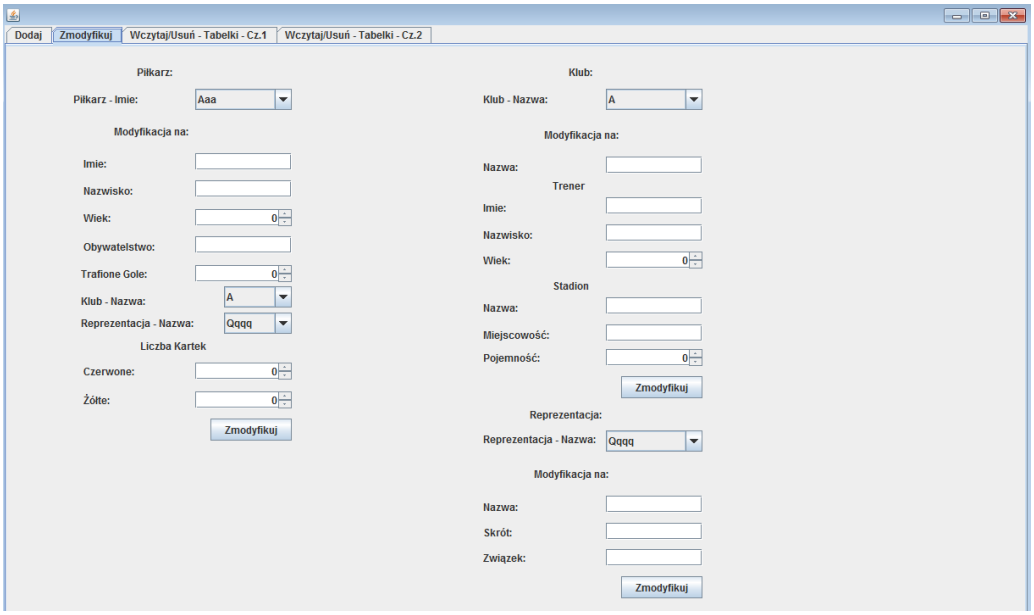
Przykładowa opcja usunięcia trenera w bazie danych. Najpierw tworzymy połączenie a następnie usuwamy dany wiersz. Warto wspomnieć że jeśli tabela jest powiązana nie jest możliwe jej usunięcie, wystąpi w takim przypadku błąd.

5. Instrukcja obsługi

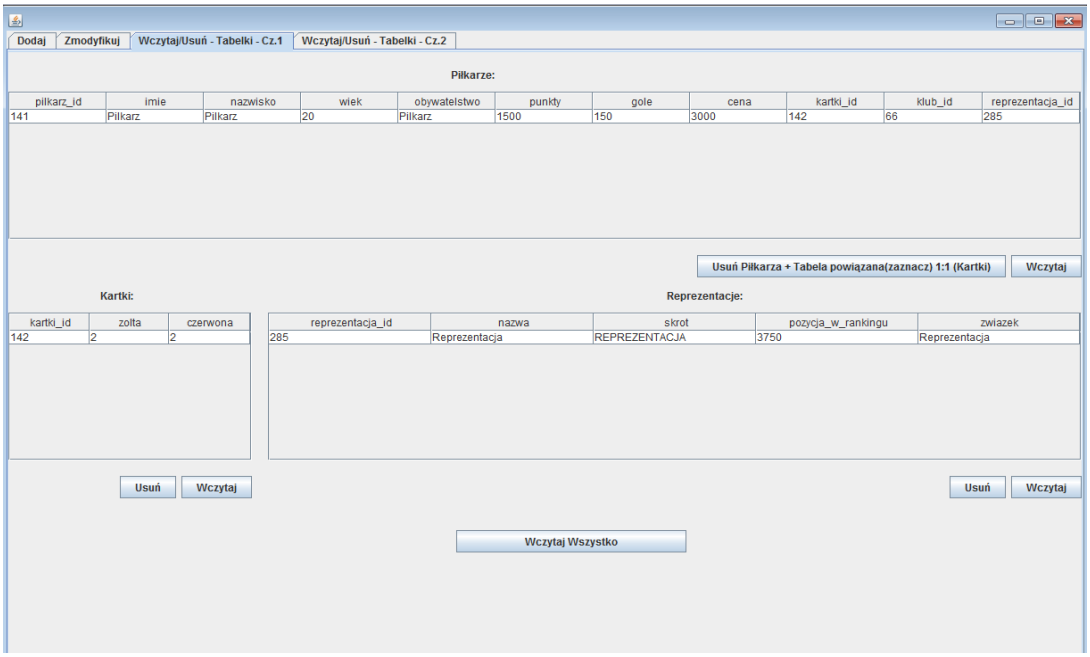
Po uruchomieniu środowiska Netbeans wystarczy kliknąć klawisz F6 (lub kliknąć zieloną strzałkę). Nasz program zostanie uruchomiony i pokaże nam się taki interfejs:



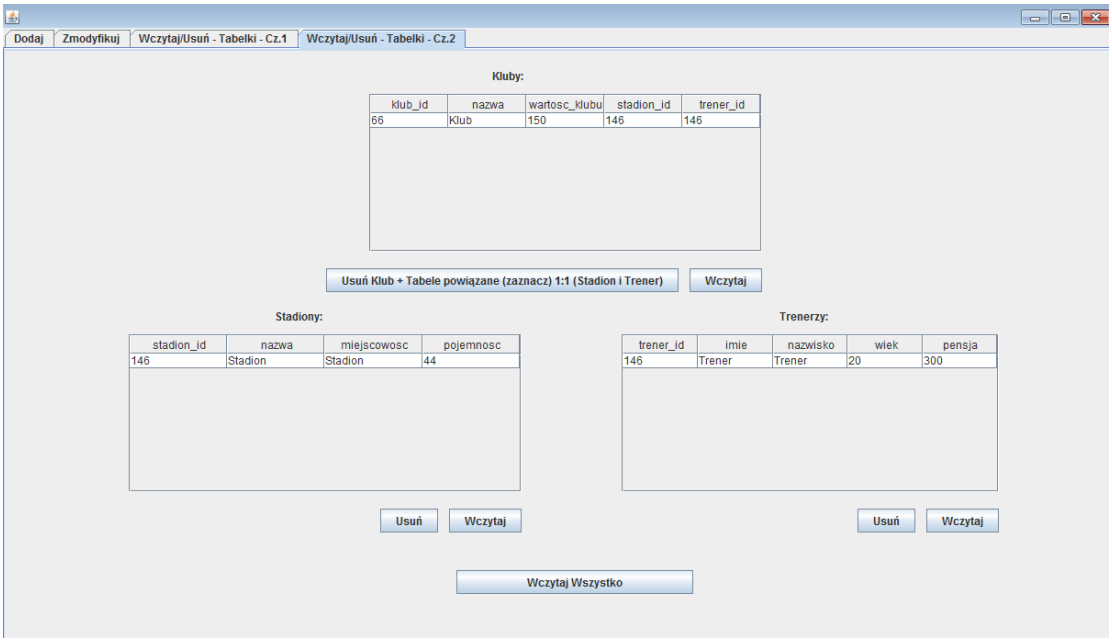
W zakładce dodaj możemy dodać wprowadzone dane do bazy danych dla tabeli Piłkarze (powiązane z kartkami), tabeli klub (powiązane z trenerem i stadionem) a także tabeli reprezentacja



W zakładce zmodyfikuj możemy zmienić wartości wprowadzone do bazy danych dla tabeli Piłkarze (powiązane z kartkami), tabeli klub (powiązane z trenerem i stadionem) a także tabeli reprezentacja



W przedostatniej zakładce możemy zobaczyć (wyświetlić) tabele w bazie danych lub usunąć je z bazy danych. Oczywiście nie jest możliwe usunięcie tabel jeśli są one powiązane. Otrzymamy wtedy komunikat o błędzie.



W ostatniej zakładce możemy również zobaczyć (wyświetlić) tabele w bazie danych lub usunąć je z bazy danych Jest to kolejną cześć przedstawionej bazy danych.

Spis treści