

	<i>Bazy Danych laboratorium</i>	<b>Laboratorium BD5</b>
--	-------------------------------------	-----------------------------

Na potrzeby zajęć zostanie wykorzystany model bazy danych opisany i zaimplementowany w ramach Laboratorium BD3.

Poza podstawowymi obiektami bazodanowymi jakimi są tabele ze swoimi strukturami i relacjami, można tworzyć i wykorzystywać w projektowaniu baz danych inne obiekty. Poniżej zostaną zaprezentowane trzy rodzaje takich obiektów: sekwencje, indeksy i perspektywy oraz dodatkowo zmienne wiązania.

## Tworzenie i zastosowanie sekwencji

Dużym problemem przy projektowaniu aplikacji bazodanowych jest zapewnienie niepowtarzalności wartości klucza głównego w tabeli. Typową sytuacją jest tabela zawierająca pewną liczbę wierszy, w której klucz główny przyjmuje kolejne wartości liczbowe (na przykład tabela faktur). Chcemy wprowadzić zdaniem *insert ....* wiersz zawierający na pozycji klucza głównego kolejną numeryczną wartość lub też zrealizować to poprzez formularz w aplikacji. Teoretycznie, najpierw trzeba by było odpytać tabelę o maksymalną wartość klucza głównego i na tej podstawie określić wartość bieżącą. Przy pomocy sekwencji ten proces można uprościć.

Sekwencja jest trwałym obiektem bazodanowym tworzonym bądź to poprzez kreator w Oracle SQL Developer bądź zdaniem *create sequence*.

Typowymi wartościami parametrów sekwencji są:

[illegible]

W wyniku takiego ustawienia parametrów można wygenerować sekwencję równoważną zdaniu SQL:

[illegible]

Najprostszym zdaniem tworzącym sekwencję jest:

```
create sequence seq_nr_faktury;
```

Tak zdefiniowana sekwencja przyjmuje standardowe wartości, które zostały przedstawione na powyższym rysunku lub:

```
create sequence seq_nr_faktury nocache;
```

, co oznacza brak cachowania kolejnych generowanych wartości klucza głównego, co może spowalniać proces ładowania danych do hurtowni danych.

W celu skasowania sekwencji należy użyć zdania *drop*, np.:

```
drop sequence seq_nr_faktury;
```

Definicję sekwencji można modyfikować zdaniem *alter sequence.....*, na przykład:

```
alter sequence seq_nr_faktury increment by 2;
```

, ale w ten sam sposób nie można zmienić frazy *start with*:<sup>1</sup>

```
alter sequence seq_nr_faktury start with 3000;
```

Wykonanie tego zdania spowoduje błąd:

```
Error report -  
ORA-02283: nie można zmienić początkowego numeru sekwencji  
02283. 00000 - "cannot alter starting sequence number"
```

Z każdą sekwencją związane są dwie wartości (pseudokolumny): *current value* ( *currval* ) i *next value* ( *nextval* ). Pierwsza z nich wskazuje ostatnią wygenerowaną wartość sekwencji (w danej sesji), a druga - wartość, która zostanie wygenerowana przy najbliższym uruchomieniu sekwencji bez względu na sesję. Wartości te można odczytać poprzez użycie zdania *select*.

```
select seq_nr_faktury.nextval from dual; -- wygenerowana jest nowa wartość  
select seq_nr_faktury.currval from dual;2 -- odczytana jest bieżąca wartość
```

Przykłady zastosowania sekwencji:

```
create sequence seq_nr_wpisu; -- utworzenie sekwencji  
  
create table testowa -- utworzenie tabeli testowej  
(  
  nr_wpisu number (5) primary key,  
  tresc varchar2 (120)  
);  
insert into testowa (nr_wpisu, tresc) -- wprowadzenie pierwszego wiersza do tabeli  
values  
(  
  seq_nr_wpisu.nextval,  
  'Pierwszy wpis testowy'  
);
```

<sup>1</sup> Można tej modyfikacji dokonać wykorzystując możliwość zmiany inkrementacji w sekwencji.

<sup>2</sup> Wartość *current value* jest określona dopiero po pierwszym odwołaniu się do sekwencji czyli wygenerowaniu wartości *next value*.

```
insert into testowa (nr_wpisu, tresc) -- wprowadzenie drugiego wiersza do tabeli
values
(
    seq_nr_wpisu.nextval,
    'Drugi wpis testowy'
);
```

`commit;` -- zatwierdzenie transakcji

Zdania `select` zwrócą poniższe wartości:

```
select * from testowa;
```

NR_WPISU	TRESC
1	Pierwszy wpis testowy
2	Drugi wpis testowy

```
select 'current value' as " ", seq_nr_wpisu.currval as "value "
from dual;
```

	value
current value	2

```
select 'next value' as " ", seq_nr_wpisu.nextval as "value "
from dual;
```

	value
next value	3

Powyższy przykład można zmodyfikować zmieniając definicję klucza głównego w tabeli:

```
alter table testowa
modify nr_wpisu default seq_nr_wpisu.nextval;
```

oraz zdanie `insert`:

```
insert into testowa (tresc) -- wprowadzenie kolejnego wiersza do tabeli
values
(
    'Trzeci wpis testowy'
);
```

Odpytanie tabeli da wynik:

NR_WPISU	TRESC
1	Pierwszy wpis testowy
2	Drugi wpis testowy
4	Trzeci wpis testowy

Numer 3 został "zużyty" w zdaniu `select` zwracającym następną wartość sekwencji w jednym z powyższych zdań.

## Tworzenie i zastosowanie indeksów

Indeks jest strukturą danych umożliwiającą szybki dostęp do wybranych wierszy tabeli bazowej w oparciu o wartości jednej lub większej liczby kolumn (klucz indeksu). Tabela bazowa nie musi być uporządkowana, to indeks powoduje, że w zbiorze wyników wiersze pojawiają się według pewnego porządku.

Indeks umożliwia przeglądanie zbioru bazowego szybciej. Optymalizator zapytań każdorazowo decyduje czy wykorzystanie gotowego indeksu przyspieszy aktualną operację i w razie potrzeby odwoła się do jego zawartości.

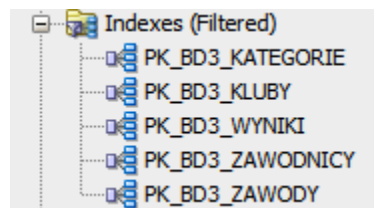
Jeżeli tabela bazowa ma przypisany klucz główny (Primary Key) system zarządzania bazą danych (SZBD) automatycznie tworzy indeks tabeli w oparciu o ten klucz. W momencie wprowadzania nowego wiersza, pierwszą czynnością wykonywaną przez SZBD jest sprawdzenie, czy klucz główny tego wiersza występuje już w indeksie danej tabeli – tak więc unikatowość wierszy w rzeczywistości jest kontrolowana na poziomie indeksu, a nie na poziomie tabeli bazowej.

Nad tworzeniem indeksów opartych o klucze główne nie mamy kontroli – SZBD utworzy je automatycznie.<sup>3</sup> Możemy jednak dodatkowo tworzyć inne indeksy pamiętając, że każdy z nich to nowa struktura.

Indeksy powodują:

- Zwiększenie bazy danych,
- Spowolnienie operacji wprowadzania, modyfikacji i usuwania wierszy z tabeli, gdyż SZBD musi zaktualizować wszystkie struktury indeksów związanych z tabelą bazową,
- Przyspieszenie pozyskiwania danych z tabel.

Strukturę indeksów można oglądać z poziomu Oracle SQL Developer rozwijając i ewentualnie filtrując folder *Indexes*:

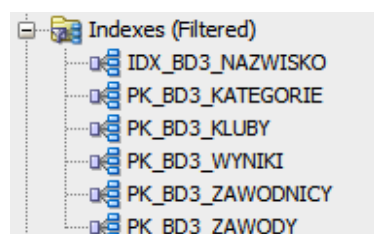


Powyższe indeksy powstały automatycznie w momencie tworzenia tabel ze zdefiniowanymi kluczami głównymi.

Istnieje możliwość zapoznania się ze szczegółowymi właściwościami danego indeksu poprzez wskazanie go i wybieranie poszczególnych zakładek (głównie Columns i SQL). Warto porównać budowę indeksu PK\_BD3\_ZAWODNICY (oparty o jedną kolumnę) oraz PK\_BD3\_WYNIKI (oparty o dwie kolumny).

Indeksy tworzy się zdaniem *create index*, np.:

```
create index idx_bd3_nazwisko on bd3_zawodnicy (nazwisko);
```



<sup>3</sup> Istnieją SZBD, na przykład Sybase, które automatycznie tworzą indeksy związane z kluczami obcymi oprócz indeksów opartych na kluczu głównym. SZBD Oracle nie posiada takiej właściwości.

Należy porównać pozycję UNIQUENESS znajdującą się w zakładce Details dla nowoutworzonego indeksu oraz dla indeksów opartych o klucz główny. Indeks oparty o nazwisko nie jest unikalny, bo nie jest kluczem głównym - nic nie stoi na przeszkodzie, aby kilku zawodników miało to samo nazwisko.

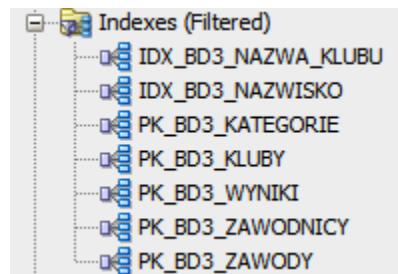
W przypadku użycia zdania:

```
select nazwisko, imie
from bd3_zawodnicy
where nazwisko like 'Kowal%'
order by nazwisko;
```

można spodziewać się, że optymalizator zapytań SQL automatycznie dobierze indeks `idx_bd3_nazwisko` w celu utworzenia zbioru wyników, co przyspiesza przetwarzanie.

Indeksy unikalne tworzone są zdaniem:

```
create unique index idx_bd3_nazwa_klubu on bd3_kluby (nazwa_klubu);
```



Poprzez indeks unikalny można kontrolować, czy nazwa klubu podana przy nowowprowadzanym klubie do ewidencji już istnieje. Można wykonać test polegający na próbie wprowadzenia jako nowego, klubu już zarejestrowanego, na przykład:

```
insert into bd3_kluby values (50, 'KU AZS WAT Warszawa');
```

Próba zakończy się niepowodzeniem, gdyż klub o tej nazwie już jest zarejestrowany.

Indeksy unikalne tworzone są automatycznie w przypadku definiowania kolumny tabeli jako *unique*:

```
alter table bd3_zawody
modify nazwa_zawodow constraint idx_bd3_nazwa_zawodow unique;
```

Usuwanie indeksów wykonuje się zdaniem *drop*:

```
drop index idx_bd3_data_urodzenia;
```

A w przypadku, gdy powstał poprzez definicję constraint (na przykład `idx_bd3_nazwa_zawodow`) należy usunąć ten constraint:

```
alter table bd3_zawody
drop constraint idx_bd3_nazwa_zawodow;
```

Pozostawia się do samodzielnej obserwacji istnienie indeksów w przypadku kasowania tabeli bazowej.

Uwagi:

1. Indeksy pełnią bardzo ważną rolę w zwiększaniu wydajności serwerów bazodanowych, dlatego są niezbędne w zarządzaniu i eksploatacji hurtowni danych,

2. Badanie wydajności baz danych nie jest oparte o ścisłe prawa czy zależności. Po części jest to proces eksperymentalny i jego efekty zależą od konfiguracji konkretnej bazy, wiedzy i doświadczenia analityka czy administratora bazy,
3. W kontekście stosowania indeksów istnieje kilka reguł, które są skuteczne w większości przypadków ich stosowania. Tworzyć należy indeksy oparte o:
  - klucze główne (tworzone automatycznie),
  - klucze obce (istotne przy łączeniu tabel),
  - kolumny występujące we frazie *where* (filtrowanie wierszy).
4. Przy badaniu sprawności wykonywania się poszczególnych zdań *select* korzysta się z planów wykonania (*executon plan*, *explain plan*).

Poniżej przedstawiony zostanie poglądowy scenariusz takiego działania:

- skasować indeks `idx_bd3_nazwisko` (jeśli istnieje),
- wykonać zdanie *select* (w celu sprawdzenia jego poprawności):

```
select nazwisko
from bd3_zawodnicy
where nazwisko = 'Kowalski';
```

- ustawić kursor myszy na terenie tego zdania i wcisnąć przycisk F10:

OBJECT_NAME	OPTIONS	CARDINALITY	COST
		5	3
BD3_ZAWODNICY	FULL	5	3

optymalizator ustalił, że to zdanie będzie się wykonywało poprzez pełen przegląd tabeli (*full*) i koszt wykonania wynosi 3,

- założyć indeks w oparciu o kolumnę `nazwisko`,

```
create index idx_bd3_nazwisko on bd3_zawodnicy (nazwisko);
```

- wygenerować plan wykonania dla zdania:

```
select nazwisko
from bd3_zawodnicy
where nazwisko = 'Kowalski';
```

OBJECT_NAME	OPTIONS	CARDINALITY	COST
		5	2
IDX_BD3_NAZWISKO	RANGE SCAN	5	2

ten *select* wcale nie będzie odczytywał danych z tabeli, gdyż nazwiska są zawarte w indeksie. Koszt wykonania jest niższy.

- skasować utworzony indeks:

```
drop index idx_bd3_nazwisko;
```

## Tworzenie i zastosowanie perspektyw

Perspektywa (view) jest tabelą wirtualną (logiczną) tworzoną zdaniem *create view...*. Ma ona własną nazwę i jest przechowywana w bazie danych z tą różnicą w stosunku do tabeli fizycznej, że brak w niej danych.

Perspektyw używa się z kilku powodów:

- Umożliwiają zapisanie często wykonywanych złożonych zapytań w strukturze bazy. Można wówczas użyć wyrażenia:

```
select * from nazwa_perspektywy;
```

To zwalnia nas z konieczności każdorazowego wpisywania całej formuły zapytania.

- Pomagają w dostosowaniu środowiska bazodanowego do indywidualnych potrzeb użytkowników lub ich grup tworząc dedykowane zapytania,
- Umożliwiają zapewnienie bezpieczeństwa danych. Zamiast nadawać użytkownikom prawa wglądu w całą strukturę tabeli, tworzona jest perspektywa zawierająca tylko te dane, które rzeczywiście są potrzebne, po czym przyznawane są prawa dostępu do wybranej perspektywy.

Perspektywy zawierają zawsze aktualne dane, gdyż ich zawartość jest tworzona każdorazowo przy wywołaniu na podstawie danych z tabel bazowych.

Perspektywa tworzona jest zdaniem *create view* według schematu:

```
create or replace view nazwa_perspektywy as
select .....
```

Przykładowo, aby utworzyć perspektywę zawierającą wybrane dane z ewidencji zawodników, można napisać poniższe zdanie:

```
create or replace view bd3_ewidencja as
select nazwisko || ' ' || imie as "Zawodnik", nazwa_klubu as "Klub"
from bd3_zawodnicy z join bd3_kluby k
on z.nr_klubu = k.nr_klubu;
```

a następnie użyć jej w prostym zapytaniu:

```
select * from bd3_ewidencja
order by "Zawodnik";
```

które da taki wynik:

⚡ Zawodnik	⚡ Klub
Adach Krzysztof	Allianz Warszawa
Adamczyk Marcin	KB Lechici Zielonka
Adamski Wojciech	KB Promyk Ciechanów
Alabrudziński Jerzy	Allianz Warszawa
Anc Michał	KB Lechici Zielonka
Anczewski Leszek	KB Trucht Warszawa
Anders Andrzej	KB Orientuz Warszawa

.....

Można w definicji perspektywy nadawać kolumnom własne nazwy, np.:

```
create or replace view bd3_ewidencja ( "Zawodnik" , "Klub" ) as
select nazwisko || ' ' || imie, nazwa_klubu
from bd3_zawodnicy z join bd3_kluby k
on z.nr_klubu = k.nr_klubu;
```

i wtedy zdanie

```
select * from bd3_ewidencja
order by "Zawodnik";
```

da taki wynik:

Zawodnik	Klub
Adach Krzysztof	Allianz Warszawa
Adamczyk Marcin	KB Lechici Zielonka
Adamski Wojciech	KB Promyk Ciechanów
Alabrudziński Jerzy	Allianz Warszawa
Anc Michał	KB Lechici Zielonka
Anczewski Leszek	KB Trucht Warszawa
Anders Andrzej	KB Orientuz Warszawa

\*\*\*\*\*

Przy tworzeniu perspektyw można korzystać z dowolnych zdań *select* – również takich, które zawierają złączenia (powyższy przykład), unie i funkcje agregujące, jak również można zagnieżdżać perspektywy.

Przykładowo:

1. *create or replace view bd3\_liczba\_zawodnikow as*  
*select count(\*) as "Liczba"*  
*from bd3\_zawodnicy;*

Wydanie polecenia:

```
select * from bd3_liczba_zawodnikow;
```

da wynik:

Liczba
771

2. *create or replace view bd3\_liczenie\_punktow as*  
*select nr\_zawodow, nr\_zawodnika, rezultat\_min, rezultat\_sek,*  
*punkty\_globalne as PktG,*  
*punkty\_kategorii as PktK*  
*from bd3\_wyniki;*  
  
*create or replace view bd3\_sumy\_pkt as*  
*select nr\_zawodnika, sum(PktG) as PktGlob, sum(PktK) as PktKat*  
*from bd3\_liczenie\_punktow*  
*group by nr\_zawodnika;*



Wydanie polecenia:

```
select *
      from bd3_sumy_pkt
     where PktGlob > 180
     order by PktGlob;
```

da wynik:

NR_ZAWODNIKA	PKTGLOBAL	PKTKAT
616	185	200
333	187	200
840	189	192
173	192	200
224	194	196
60	198	200

W celu usunięcia perspektywy z bazy należy użyć zdania *drop view nazwa\_perspektywy*:

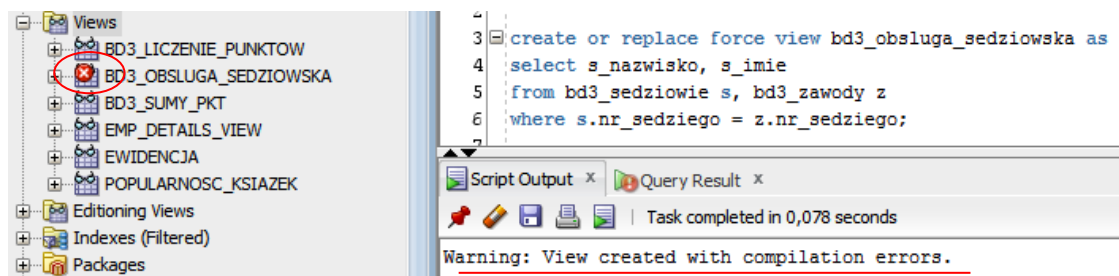
Np.: *drop view bd3\_ewidencja*;

Uwagi:

1. W Oracle SQL Developer można obserwować tworzenie i usuwanie perspektyw wybierając folder Views,
2. Należy zauważyć, że definicje perspektyw są zapamiętane w bazie po zamknięciu sesji (na stałe).
3. Tworzona perspektywa może mieć nadaną właściwość *force*:

```
create or replace force view bd3_obsługa_sedziowska as
select s_nazwisko, s_imie
from bd3_sedziowie s, bd3_zawody z
where s.nr_sedziego = z.nr_sedziego;
```

W schemacie brak jest tabeli BD3\_SEDZIOWIE oraz brak jest kolumny nr\_sedziego w tabeli BD3\_ZAWODY. Klauzula *force* spowoduje, że perspektywa zostanie założona, chociaż będzie wykazany błąd.



Właściwość ta może być wykorzystana w pracach projektowych (na przykład zespołowych), w trakcie których można projektować fragmenty systemu w oparciu o istniejące nazwy perspektyw, chociaż brak jest implementacji ostatecznych definicji tabel.

4. W przypadku tworzenia perspektywy na podstawie zdania *select* zawierającego funkcję agregującą warto zwrócić uwagę na zachowanie się agregowanej wartości. Załóżmy, że opracowane jest zdanie *select* z funkcją agregującą, na przykład:

```
select nazwa_klubu "Klub", sum ( punkty_globalne ) "Punktacja"
from bd3_zawodnicy z join bd3_kluby k on z.nr_klubu = k.nr_klubu
join bd3_wyniki w on z.nr_zawodnika = w.nr_zawodnika
group by nazwa_klubu;
```

Chcąc, otrzymany na podstawie tego zdania, zbiór wynikowy ograniczyć należy zastosować frazę *having*:

```
.....
having sum ( punkty_globalne ) > 1000
```

Próba użycia do tego filtrowania frazy *where* zakończy się niepowodzeniem:

```
SQL Error: ORA-00934: funkcja grupowa nie jest tutaj dozwolona
00934. 00000 - "group function is not allowed here"
```

Po zbudowaniu na podstawie tego zdania perspektywy:

```
create or replace view bd3_punktacja_klubowa ( klub, punktacja ) as
select nazwa_klubu, sum ( punkty_globalne )
from bd3_zawodnicy z join bd3_kluby k on z.nr_klubu = k.nr_klubu
join bd3_wyniki w on z.nr_zawodnika = w.nr_zawodnika
group by nazwa_klubu;
```

i próbie użycia frazy *having* przy jej uruchomieniu:

```
select * from bd3_punktacja_klubowa
having punktacja > 1000
order by punktacja desc;
```

wykazany zostanie błąd:

```
SQL Error: ORA-00979: to nie jest wyrażenie GROUP BY
00979. 00000 - "not a GROUP BY expression"
```

, natomiast użycie frazy *where* zamiast *having* zakończy się powodzeniem:

KLUB	PUNKTACJA
KB Trucht Warszawa	2607
KB Gymnasion Warszawa	1763
Allianz Warszawa	1425

Kolumna będąca zbiorem agregatów w zdaniu *select* została niejawnie przekształcona w kolumnę będącą zbiorem wielkości skalarnych.

5. Perspektyw można używać w zdaniach DML (*insert*, *update*, *delete*), ale w ograniczonym zakresie. Do najważniejszych ograniczeń należą:
- nie może być oparta na wielu tabelach,
  - nie może zawierać klauzuli *distinct*,
  - nie może zawierać funkcji grupujących ani klauzuli *group by*,
  - nie może zawierać wyrażeń w kolumnie.

## Zastosowanie zmiennych wiązania w języku SQL

Zmienne wiązania (*bind variable*) mają, między innymi, zastosowanie w przekazywaniu danych między środowiskami programistycznymi, na przykład między językiem SQL i java, php, C itp. Innym zastosowaniem jest wprowadzanie danych do zdań SQL w pracy interaktywnej w Oracle SQL Developer lub SQL Plus.

Założmy, że chcemy testować zdanie *select* postaci:

```
select nazwisko || ' ' || imie "Zawodnik", nazwa_klubu "Klub"
from bd3_zawodnicy z join bd3_kluby k
on z.nr_klubu = k.nr_klubu
where k.nr_klubu = xxx ;      -- xxx - numer klubu wprowadzany
                             --          interaktywnie
```

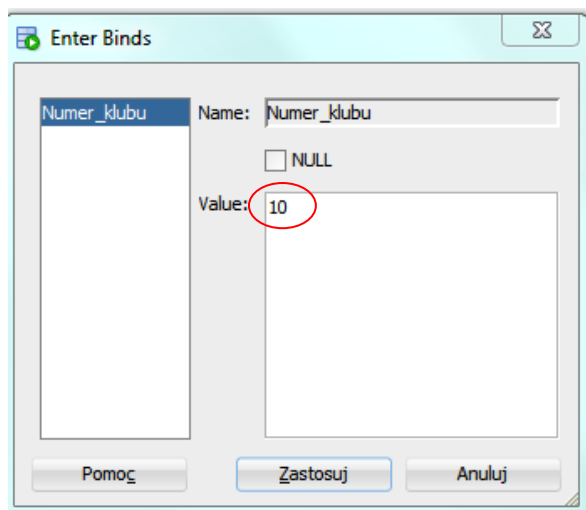
Oczywiście, że nic nie stoi na przeszkodzie, aby zmieniać interaktywnie wartość xxx przez nadpisanie. Ale można również to zdanie sparаметryzować, na przykład:

```
select nazwisko || ' ' || imie "Zawodnik", nazwa_klubu "Klub"
from bd3_zawodnicy z join bd3_kluby k
on z.nr_klubu = k.nr_klubu
where k.nr_klubu = :Numer_klubu;
```

gdzie zmienna Numer\_klubu (poprzedzona znakiem :) jest zmienną wiązania.

W tym przypadku zmienna ta nie musi być deklarowana oraz może mieć dowolną nazwę, na przykład :NR lub :Podaj\_Numer\_Klubu.

Po uruchomieniu tak skonstruowanego zdania w Oracle SQL Developer (F5) wyświetlony zostanie formularz:



, na którym należy w polu Value wprowadzić konkretną wartość, w tym przypadku numer klubu, zawodników którego ma dotyczyć tworzony zbiór wynikowy.

Jednocześnie można definiować wiele zmiennych wiązania, na przykład uruchomienie zdania:

```
select nazwisko || ' ' || imie "Zawodnik", nazwa_klubu "Klub"
from bd3_zawodnicy z join bd3_kluby k
on z.nr_klubu = k.nr_klubu
where z.nr_klubu = :Numer_klubu
and data_urodzenia between :min_Data_urodzenia and :max_Data_urodzenia;
```

spowoduje wyświetlenia formularza, na którym należy wprowadzić wartości dla wszystkich zmiennych wiązania. Przykładowo powyższe zdanie dla:

```
min_Data_urodzenia = 1973/01/01      -- bez apostrofów
max_Data_urodzenia = 1973/01/31
Numer_klubu = 10
```

utworzy zbiór wynikowy:

Zawodnik	Klub
Mioduszewski Mateusz	AZS Uniwersytet Warszawski

, a zdanie:

```
select nazwisko || ' ' || imie "Zawodnik", nazwa_klubu "Klub"
from bd3_zawodnicy z join bd3_kluby k
on z.nr_klubu = k.nr_klubu
where z.nr_klubu = :Numer_klubu
and nazwisko like :bind_nazwisko;
```

dla danych:

```
Numer_klubu = 10
bind_nazwisko = %Ma%      -- bez apostrofów
```

utworzy zbiór:

Zawodnik	Klub
Mazurczyk Rafał	AZS Uniwersytet Warszawski
Marciniak Andrzej	AZS Uniwersytet Warszawski
Marcinkowski Tomasz	AZS Uniwersytet Warszawski

Uwaga:

Według tej samej zasady odbywa się przekazywanie danych w aplikacjach bazodanowych. Pola na formularzu aplikacji są zmiennymi wiązania i po jego wypełnieniu przenoszone są do zdania SQL, na przykład:

```
insert into ewidencja (nazwisko, imie, pesel, adres_zamieszkania)
values (:P10_NAZWISKO, :P10_IMIE, :P10_PESEL, :P10_ADRES);
```

gdzie zmienne z prefixem P10 są nazwami pól na formularzu aplikacji.

### Zadania do samodzielnego wykonania:

1. Opracować perspektywę *bd3\_liczba\_zawodnikow* zliczającą zarejestrowanych zawodników w rozbiciu na płeć, a następnie na jej podstawie obliczyć liczbę zaewidencjonowanych zawodniczek.
2. Opracować perspektywę *bd3\_ewidencja* zawierającą kolumny:

*Nazwisko, Klub, Kategoria, Wynik, Zawody, Data zawodów*

przy czym:

Nazwisko jest złożeniem kolumn *nazwisko* i *imie*,  
Klub to *nazwa\_klubu*,  
Kategoria to *nazwa\_kategorii*,  
Wynik to złożenie kolumn *rezultat\_min* i *rezultat\_sek* i przedstawienie jej w formacie mm:ss, (wykorzystać funkcję *to\_char* lub *cast*),  
Zawody to kolumna *nazwa\_zawodow*,  
Data Zawodów to kolumna *data\_zawodow*.

Opracować zdanie SQL, które na podstawie tej perspektywy będzie generować listę osiągnięć podanego tylko z nazwiska zawodnika porządkując zbiór malejąco według daty zawodów.

3. Założyć tabelę *BD3\_KLASYFIKACJA\_GEN* przy pomocy konstrukcji *create table as select...* przechowującą nr klubu, sumę zdobytych punktów w klasyfikacji generalnej, liczbę zawodników każdego klubu startujących w zawodach oraz liczbę zawodników zaewidencjonowanych w każdym klubie. Na jej podstawie opracować perspektywę generującą raport pokazujący klasyfikację klubów, w którym są uwzględnione tylko te kluby, które zdobyły punkty w tej klasyfikacji zawierający nazwę klubu, sumę zdobytych punktów w klasyfikacji globalnej oraz liczbę startujących zawodników.  
Należy wykorzystać technikę złączeń zewnętrznych oraz odpowiednio zastosować argument funkcji *count*.
4. Założyć sekwencję umożliwiającą wprowadzanie nowych wierszy do tabeli *BD3\_ZAWODY* przy uwzględnieniu już istniejących.
5. Założyć indeks zwiększający wydajność serwera dla wybranych kluczy obcych oraz dla wybranych kolumn często stosowanych przy filtrowaniu bazy (na przykład *nazwisko* i *imie*).