

	<i><b>Bazy Danych laboratorium</b></i>	<b>Laboratorium BD4</b>
--	--	-----------------------------

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

## Wykonywanie obliczeń w zdaniach SQL

W zdaniach SQL można wykonywać proste operacje arytmetyczne na wartościach kolumn i na stałych używając do tego celu klasycznych operatorów działań: +, -, \*, / oraz ().

Przykładowo:

Zdanie:

```
select 125 + 15 / 50 - 3 as "I przykład",
       (125 + 15) / 50 - 3 as "II przykład",
       (125 + 15) / (50 - 3) as "III przykład"
from dual;
```

da poniższe wyniki:

I przykład	II przykład	III przykład
122,3	-0,2	2,97872340425531914893617021276595744681

Należy zwrócić uwagę na konstrukcję zdania *select*. W podstawowej swej formie jego definicja zawiera frazę *from*: *select ..... from tabela*. Implementacja SQL w środowisku Oracle nie dopuszcza skróconej formy *select ..... bez frazy from*, w przypadku kiedy dane lub obliczenia nie pochodzą z żadnej tabeli<sup>1</sup>.

Jedną z wielu funkcji operujących na danych liczbowych jest funkcja zaokrąglenia, której ogólna postać jest następująca:

*round* (zmienna liczbowo, n),

gdzie : zmienna liczbowo może być nazwą kolumny tabeli lub wartością,  
n – dokładność zaokrąglenia.

Przykładowo powyższe zdanie SQL z tą funkcją (domyślnie n=0):

```
select round (125 + 15 / 50 - 3) as "I przykład",
       round ((125 + 15) / 50 - 3) as "II przykład",
       round ((125 + 15) / (50 - 3)) as "III przykład"
from dual;
```

da wynik:

I przykład	II przykład	III przykład
122	0	3

<sup>1</sup> W innych systemach zarządzania bazami danych (np. Sybase) nie jest to konieczne i frazę *from* można pominąć. Aby być w zgodzie z klasyczną formą zdania *select* wprowadza się fikcyjną tabelę (*dual* dla Oracle, *dummy* dla Sybase), której można używać w formie: *select ....from dual*.

Zdanie:

```
select round (25.123456 , 2) as "Round to 2",
       round (25.123456 , 4) as "Round to 4",
       round (25.123456 , 0) as "Round to 0",
       round (25.123456 , -1) as "Round to -1"
from dual;
```

da poniższe wyniki:

Round to 2	Round to 4	Round to 0	Round to -1
25,12	25,1235	25	30

Oczywiście obliczeń można dokonywać także na danych pochodzących z tabel. Przykładowo poniższe zdanie generuje zbiór wynikowy, w którym obliczono wiek zawodników:

```
select imie || ' ' || nazwisko as "Zawodnik", nazwa_klubu as "Klub",
       round (sysdate - data_urodzenia) as "Wiek"
from bd3_zawodnicy z, bd3_kluby k
where z.nr_klubu = k.nr_klubu and z.nr_klubu = 3
order by "Wiek" desc;
```

Zawodnik	Klub	Wiek
Edmund Werthein	KB Gymnasion Warszawa	33807
Bożena Gradus	KB Gymnasion Warszawa	33163
Wojciech Bielowicki	KB Gymnasion Warszawa	31844
Alek Borowski	KB Orientuz Warszawa	31226
Adam Gorzkowski	KB Lechici Zielonka	31156
Jan Maliszewski	Bielański KB Warszawa	31115
Jarosław Żelazko	KB Gymnasion Warszawa	31078

Powyższy przykład ma tę wadę, że wiek obliczony jest w dniach po odjęciu dwóch dat od siebie. Nic nie stoi na przeszkodzie, aby prostym działaniem arytmetycznym przeliczyć dni na lata. Natomiast zaletą jest użycie zmiennej *sysdate*, która jako zmienna globalna zawiera bieżącą datę na serwerze bazodanowym.

## Wykonywanie operacji na łańcuchach znakowych

Istnieje szereg funkcji pozwalających na przetwarzanie łańcuchów tekstowych.

Jedną z nich jest omówiony w Laboratorium BD2 operator konkatencji (łączenia łańcuchów), którego zapis wygląda tak:

kolumna\_1 || kolumna\_2 || '...łańcuch znaków...' || .....

Podobną rolę pełni funkcja *concat*, której ogólna postać wygląda tak:

*concat* (ciąg\_znaków\_1, ciąg\_znaków\_2)

Wadą tej funkcji jest to, że jest dwuargumentowa. Chcąc jej użyć do łączenia imienia z nazwiskiem należy zastosować zagnieżdżenie funkcji:

..... *concat* ( *concat* (imie, ' '), nazwisko ) .....

Najbardziej popularnymi funkcjami przetwarzającymi łańcuchy znakowe są funkcje:

- *upper*,
- *lower*,
- *substr*,
- *initcap*.

### Funkcje *upper* i *lower*

Funkcje *upper* i *lower* są funkcjami zamieniającymi cały łańcuch znaków odpowiednio na duże i małe litery:

```
select lower ( imie ) as "Imię", upper ( nazwisko ) as "Nazwisko"
from bd3_zawodnicy;
```

Imię	Nazwisko
piotr	KUCZKOWSKI
aleksander	CIEŚLAK
tadeusz	STRZAŁKOWSKI
jan	KOWALSKI
grzegorz	GRABOWSKI
petter	CIEŚLAK

Użycie tych funkcji ma dwa praktyczne zastosowania.

Pierwsze to wygląd opracowywanych raportów. Na powyższym przykładzie widać, że końcowy raport będzie zawierał nazwiska zapisane dużymi literami, chociaż w bazie tak nie jest.

Drugie zastosowanie dotyczy metod przeszukiwania bazy i jej modyfikacji.<sup>2</sup> Jeśli baza danych w momencie tworzenia zostanie ustawiona na rozróżnianie wielkości liter w danych, to nie bez znaczenia jest czy warunek filtrowania będzie napisany tak:

```
.....
where imie = 'stefan'
```

czy też tak:

```
.....
where imie = 'Stefan'
```

W takim przypadku, aby ustrzec się przed błędami wyszukiwania bezpiecznie jest użyć jednej z tych funkcji w warunku wyboru, np.:

```
select lower ( imie ) as "Imię", upper ( nazwisko ) as "Nazwisko"
from bd3_zawodnicy z, bd3_kluby k
where z.nr_klubu = k.nr_klubu and upper ( imie ) = 'STEFAN';
```

Funkcja *upper* we frazie *where* powoduje, że wszystkie imiona z tabeli BD3\_ZAWODNICY są zamieniane na duże litery i porównywane z napisem 'STEFAN'. W przypadku zgodności wybrane wiersze przechodzą do zbioru wyników, w którym imiona są zamieniane na małe litery zgodnie z listą we frazie *select*.

To rozwiązanie jest stosowane bardzo często w aplikacjach bazodanowych, w których dane pochodzą z formularzy wypełnianych przez użytkownika. Aby zachować jednolitość bazy można w zdaniach *insert*, *update* stosować przekształcenia łańcuchów znaków.

<sup>2</sup> W bazie oraclovej na serwerze *city.wsisiz.edu.pl* ten przypadek zachodzi czyli 'abacki' nie oznacza tego samego co 'Abacki' lub 'ABACKI' (*case sensitivity*). Można spotkać inne instalacje, w których ta zależność nie zachodzi.

Np.

```
insert into tabela (.....)
values
(
    upper ( :zm_1 ),
    upper ( :zm_2 ),
    .....
);
lub:

update tabela
set kolumna_N = lower ( :zm_N )
where kolumna_M = lower ( :zm_M );
```

Uwaga:

Nazwy zmiennych poprzedzone znakiem dwukropka oznaczają zmienne wiązania (*bind variable*), które służą do przekazywania danych między środowiskami programistycznymi, takimi jak java, php, c#, html i środowiskiem bazodanowym czyli np. SQL czy PL/SQL. Typowym przykładem są formularze aplikacji, z których po ich wypełnieniu wartość pól łączona jest ze zdaniem SQL (*parsowanie*) w celu wprowadzenia danych do tabeli lub przeszukiwania bazy.

### Funkcja *substr*

Funkcja *substr* wycina fragment z podanego łańcucha znaków. Ogólna postać tej funkcji wygląda tak:

*substr* (łańcuch znaków, pozycja startowa, liczba znaków)

Przykładowo chcąc utworzyć trzyliterowe skróty nazwisk zawodników można użyć konstrukcji:

```
select nazwisko as " Nazwisko", substr (nazwisko, 1, 3) as "Identyfikator"
from bd3_zawodnicy;
```

która da poniższy zbiór wyników:

Nazwisko	Identyfikator
Kuczkowski	Kuc
Cieślak	Cie
Strzałkowski	Str
Kowalski	Kow
Grabowski	Gra
Cieślak	Cie

### Funkcja *initcap*

Funkcja *initcap* zamienia podany argument w ten sposób, że każdy pierwszy znak słowa zamienia na dużą literę, a pozostałe czyni małymi. Ogólna postać wygląda tak:

*initcap* ( ciąg\_znaków )

Przykładowo zdanie:

```
select initcap ( 'ADAM ABACKI' ) as "Nazwisko" from dual;
```

da poniższy efekt:

Nazwisko
Adam Abacki

Uwaga:

Istnieje szereg innych funkcji wzbogacających specyfikację języka SQL i PL/SQL. Szczegółowy opis można znaleźć w dokumentacji Oracle (lub innych serwerów bazodanowych). Dodatkowo warto zapoznać się z zawartością serwisów [Tech on the Net](#) lub [Oracle Tutorials](#).

## Wykonywanie operacji na datach

Data w systemach bazodanowych przechowywana jest jako wartość liczbowa mieszana. Część całkowita określa liczbę dni jaka upłynęła od pewnego dalekiego w przeszłości dnia (np. dla Oracle jest to dzień 1 stycznia 4712 p.n.e.), a część ułamkowa, jeśli występuje, wskazuje na czas w ramach jednej doby (np. wartość 0,5 określa południe) z dokładnością do milisekund.

Drugim ważnym aspektem związanym z datami jest sposób wprowadzania dat do bazy danych.

Założmy, że chcemy wprowadzić do tabeli BD3\_ZAWODY informację o kolejnych zawodach:

```
insert into bd3_zawody ( data_zawodow, nazwa_zawodow, nr_zawodow )
values ( ' 24/02/2018 ', 'Grand Prix Warszawy', 5);
```

W momencie uruchomienia tego zdania pojawi się błąd:<sup>3</sup>

```
Error starting at line : 1 in command -
insert into bd3_zawody (data_zawodow, nazwa_zawodow, nr_zawodow)
values ('24/02/2018', 'Grand Prix Warszawy', 5)
Error report -
ORA-01830: wzorzec formatu daty kończy się przed konwersją całego napisu wejściowego
```

wskazujący na zły format wprowadzanej daty.

Jak zatem postępować chcąc wprowadzać dane typu data do bazy danych?

Po pierwsze trzeba się dowiedzieć, jaki format daty jest przyjęty za standardowy na konkretnym serwerze. Format ten ustala administrator serwera w momencie jego instalowania. Aby to zrobić należy odczytać z serwera datę systemową.

Poniżej zostanie zaprezentowane zdanie pokazujące użycie dwóch funkcji standardowych określających zmienne systemowe przechowujące datę i czas systemowy.

```
select sysdate as "Data systemowa",
       systimestamp as "Czas systemowy dokładny"
from dual;
```

Data systemowa	Czas systemowy dokładny
18/09/27	18/09/27 15:28:51,463109000 +02:00

<sup>3</sup> Należy zwrócić uwagę, że daty, tak samo jak wartości znakowe, opatrzone są apostrofami jako znakami ograniczającymi.

Po drugie do wprowadzania dat używać odczytanego formatu. W tym przypadku jest to format określany jako 'YYYY/MM/DD' mimo, że na powyższym rysunku widnieje 'YY/MM/DD'. Zostanie to wyjaśnione w dalszej części materiału.

Zatem zdanie wprowadzające nową pozycję do tabeli BD3\_ZAWODY powinno wyglądać tak:

```
insert into bd3_zawody ( data_zawodow, nazwa_zawodow, nr_zawodow )
values ( ' 2018/02/24 ', 'Grand Prix Warszawy', 5);
```

Na datach można wykonywać operacje arytmetyczne, ale w ograniczonym zakresie.

Typowymi i najczęściej spotykanymi działaniami są:

date + integer	- dodanie do daty pewnej liczby dni,
date - integer	- odjęcie od daty pewnej liczby dni,
date – date	- obliczenie liczby dni między dwiema datami,

Przykładowo zdanie:

```
select sysdate      as "Data systemowa",
       sysdate + 5   as "Termin końcowy"
from dual;
```

daje wynik:

↕ Data systemowa	↕ Termin końcowy
18/09/27	18/10/02

ale wykonanie zdania:

```
select ' 2018/09/26 ' + 5 from dual;
```

jest niemożliwe:

```
Error starting at line : 1 in command -
select '2018/09/26' + 5 from dual
Error report -
ORA-01722: niepoprawna liczba
```

Dlaczego?

Zapis '2018/09/26' nie jest traktowany jako data, jeśli nie jest odczytywany z tabeli. W powyższym przykładzie jest on traktowany jako ciąg znaków i nie można go używać do działań arytmetycznych.

Natomiast możliwe jest używanie tego zapisu w zdaniach *insert*, *update*, *delete* i we frazie *where*, np.:

```
.....
where data_urodzenia between '1986/01/01' and '1986/12/31'
.....
```

Chcąc użyć daty w takiej postaci do bezpośrednich obliczeń należy najpierw ją w sposób jawny przekonwertować czyli użyć funkcji konwersji.

## Funkcje konwersji

W celu jawnej zmiany typu przetwarzanych danych stosuje się funkcje konwersji. Standard Oracla przewiduje między innymi takie funkcje konwersji: *to\_date*, *to\_char*, *to\_number*, *to\_timestamp* itp. Niektóre z nich zostaną omówione poniżej.

### *to\_date*:

Podstawową funkcją konwersji jest funkcja *to\_date*, która zamienia datę w postaci napisu na datę w postaci wyrażenia. Ogólna postać tej funkcji wygląda tak, jak poniżej:

*to\_date* ( napis\_będący\_datą , format )

gdzie:

napis\_będący\_datą - ciąg znaków, który ma być przekonwertowany na datę,  
format - informacja dla interpretera, w jakiej postaci jest przedstawiony napis\_będący\_datą,  
jest on budowany z znaków symbolizujących lata (YYYY), miesiące (MM lub MON lub MONTH) i dni (DD).

Na przykład zdanie:

```
select  to_date ( '2018.09.26', 'YYYY.MM.DD' ) + 5 as "Termin I",
        to_date ( '26-WRZ-2018', 'DD-MON-YYYY' ) + 10 as "Termin II",
        to_date ( '26-WRZESIEŃ-2018', 'DD-MONTH-YYYY' ) + 15 as "Koniec"
from dual;
```

daje wynik:

Termin I	Termin II	Koniec
18/10/01	18/10/06	18/10/11

Uwaga:

Mimo, że nastąpiła konwersja daty do postaci YYYY/MM/DD, sposób jej prezentacji wygląda jako YY/MM/DD. Aby dociec dlaczego, należy odczytać parametry NLS (*National Language Support*) serwera bazodanowego. Są to parametry określające między innymi język narodowy, symbol waluty narodowej i formaty daty i czasu ustawione dla danego serwera. Wykonać to można zdaniem:

```
select parameter, value from v$nls_parameters;
```

Widać, że językiem jest polski czyli polskie znaki będą prawidłowo interpretowane, a data i czas mają format RR/MM/DD. Format RR określa rok w postaci dwuznakowej i różni się od YY.

Aby się o tym przekonać można do tabeli BD3\_ZAWODY wprowadzić dane przy pomocy poniższych zdań:

```
insert into bd3_zawody ( data_zawodow, nazwa_zawodow, nr_zawodow )
values (to_date ( '98-02-24', 'YY-MM-DD' ), 'Grand Prix Warszawy', 5);
insert into bd3_zawody ( data_zawodow, nazwa_zawodow, nr_zawodow )
values (to_date ( '98-02-24', 'RR-MM-DD' ), 'Grand Prix Warszawy', 6);
```

Po odpytaniu tabeli zdaniem *select \* from bd3\_zawody* otrzymamy wyniki:

NR_ZAWODOW	NAZWA_ZAWODOW	DATA_ZAWODOW	PODTYTUL
1	Cztery Pory Roku - Jesień	10/10/22	(null)
2	Cztery Pory Roku - Zima	11/02/19	(null)
3	Cztery Pory Roku - Wiosna	11/05/21	(null)
4	Cztery Pory Roku - Lato	11/09/03	im.S.Dankowskiego
5	Grand Prix Warszawy	98/02/24	(null)
6	Grand Prix Warszawy	98/02/24	(null)

Pozornie obie daty w nowowprowadzonych wierszach wyglądają tak samo. Ale jest różnica.

Każdy użytkownik może chwilowo zmienić parametry NLS w ramach swojej bieżącej sesji. Należy użyć na przykład takiego zdania:

```
alter session
set nls_date_format = 'YYYY/MM/DD';
```

Po jego wykonaniu ponowne odpytanie tabeli BD3\_ZAWODY da inny wynik:

NR_ZAWODOW	NAZWA_ZAWODOW	DATA_ZAWODOW	PODTYTUL
1	Cztery Pory Roku – Jesień	2010/10/22	(null)
2	Cztery Pory Roku – Zima	2011/02/19	(null)
3	Cztery Pory Roku – Wiosna	2011/05/21	(null)
4	Cztery Pory Roku – Lato	2011/09/03	im.S.Dankowskiego
5	Grand Prix Warszawy	2098/02/24	(null)
6	Grand Prix Warszawy	1998/02/24	(null)

W przypadku, gdy do tabeli wprowadzana jest data, w której rok jest określony na dwóch pozycjach to:

- jeśli jest to format YY i rok  $\geq 50$  to jest on umieszczany w XXI wieku,
- jeśli jest to format RR i rok  $\geq 50$  to jest on umieszczany w XX wieku,
- jeśli rok  $< 50$  to bez względu na format (YY lub RR) jest on umieszczany w XXI wieku.

Wnioski końcowe:

1. Bardzo często administratorzy baz danych mają do czynienia z sytuacją, że muszą wczytywać do bazy danych (hurtowni) dane z plików pochodzących z różnych systemów (na przykład w korporacjach globalnych). Formaty dat mogą się różnić między źródłami danych i serwerami bazodanowymi. Powyższe rozważania mogą tę czynność znakomicie przyspieszyć, gdyż łatwo sobie wyobrazić jakim nakładem opatrzona jest modyfikacja pliku płaskiego zawierającego, na przykład, jeden milion wierszy z datą niezgodną z docelowym formatem.

2. Ustawienie parametrów NLS przy pomocy konstrukcji *alter session ...* jest obowiązujące do czasu zamknięcia sesji. Po ponownym zalogowaniu się do schematu obowiązują parametry ustawione na serwerze, a nie w sesji.

#### extract:

Jest to funkcja dokonująca ekstrakcji jednego z elementów daty lub czasu, takich jak rok, miesiąc, dzień, godzina, minuta czy sekunda. Nazwy atrybutów tej funkcji są synonimami ekstrahowanych wielkości (w języku angielskim):

Zdanie

```
select extract ( year from sysdate ) as Rok
      , extract ( month from sysdate ) as Miesiąc
      , extract ( day from sysdate ) as Dzień
from dual;
```

zwraca wyniki:

ROK	MIESIĄC	DZIEŃ
2018	9	28

A zdanie:

```
select extract ( hour from systimestamp ) as Godzina
      , extract ( minute from systimestamp ) as Minuta
      , extract ( second from systimestamp ) as Sekunda
from dual;
```



zwraca wyniki:

GODZINA	MINUTA	SEKUNDA
18	47	38,928741

Oczywiście zamiast zmiennej globalnej *sysdate* czy *systimestamp* można użyć dowolnej innej wartości będącej datą lub czasem lub kolumny tabeli typu *date* lub *timestamp*:

Na przykład zdanie:

```
select nr_zawodnika
      , extract ( year from sysdate ) as "Bieżący rok"
      , extract ( year from data_urodzenia ) as "Rok urodzenia"
      , extract ( year from sysdate ) - extract ( year from data_urodzenia ) as "Wiek"
from bd3_zawodnicy;
```

da wynik:

NR_ZAWODNIKA	Bieżący rok	Rok urodzenia	Wiek
263	2018	1975	43
264	2018	1977	41
265	2018	1966	52
266	2018	1970	48

.....

Uwaga:

W środowisku Oracle nie jest możliwe użycie poniższej konstrukcji:

```
select nr_zawodnika
      , extract ( year from sysdate ) as "Bieżący rok"
      , extract ( year from data_urodzenia ) as "Rok urodzenia"
      , "Bieżący rok" - "Rok urodzenia" as "Wiek"
from bd3_zawodnicy;
```

Nie można używać aliasów do obliczeń bezpośrednio w liście *select*, można ich używać tylko we frazie *order by*. Powyższy problem rozwiązuje się przy pomocy innej konstrukcji, która będzie omówiona w dalszej części materiału.

**to\_char:**

Funkcja ta konwertuje wartości typu *number*, *date* lub *timestamp* do postaci znakowej. Najczęściej stosuje się ją do innej prezentacji danych z tabeli, na przykład, wymaganej na tworzonych raportach. Ogólna postać tej funkcji wygląda tak, jak poniżej:

*to\_char* ( wartość\_numeryczna\_lub data/czas , format )

gdzie:

wartość\_numeryczna\_lub data/czas - kolumna tabeli, zmienna lub konkretna wartość typu *number*, *date* lub *timestamp*,

format - wzorec docelowej postaci przekonwertowanych danych, do jego zdefiniowania w przypadku dat używa się takich symboli jak: YYYY lub YY, Q, MM, MONTH, MON, DD, DY, DAY.

Poniżej zostanie zaprezentowane użycie tej funkcji w stosunku do dat. Konwersja danych numerycznych nie będzie omawiana w tym materiale.

Przykładowo zdanie:

```
select nazwisko, data_urodzenia
      , to_char ( data_urodzenia, 'DAY DD.MM.YYYY') as "Data urodzenia"
from bd3_zawodnicy;
```

da wynik:

NAZWISKO	DATA_URODZENIA	Data urodzenia
Kuczkowski	75/05/26	PONIEDZIAŁEK 26.05.1975
Cieślak	77/07/22	PIĄTEK 22.07.1977
Strzałkowski	66/06/28	WTOREK 28.06.1966
Kowalski	70/09/17	CZWARTEK 17.09.1970
Grabowski	79/10/11	CZWARTEK 11.10.1979

.....

Funkcje konwersji, podobnie jak inne funkcje, mogą podlegać zagnieżdżaniu. Załóżmy, że chcemy datę w postaci napisu przekonwertować na inną postać napisu. Wykonanie zdania:

```
select to_char ( '24-02-1996', 'DD-MM-YYYY') as "Data urodzenia"
from dual;
```

zakończy się niepowodzeniem, gdyż wprowadzana data nie jest w akceptowanym formacie:

```
select to_char ( '24-02-1996', 'DD-MM-YYYY') as "Data urodzenia"
from dual
Error report -
ORA-01722: niepoprawna liczba
```

Należy najpierw z napisu zrobić datę, a następnie powtórnie dokonać konwersji według innego formatu, na przykład:

```
select to_char ( to_date ( '24-02-1996', 'DD-MM-YYYY'),
      'fm DY DD MONTH YYYY') as "Data urodzenia"
from dual;
```

Data urodzenia
SO 24 LUTY 1996

Prefix *fm* służy do wskazania, aby zbędne spacje w przekonwertowanej wartości zostały usunięte.

Uwaga:

Konwersja przy pomocy funkcji *to\_char* w celu prezentacji w zakładanej formie danych numerycznych czy też dat straciła na znaczeniu w języku SQL z powodu powstania szeregu wyspecjalizowanych programów do tworzenia raportów. Mają one zaimplementowane, co najmniej, takie same funkcjonalności związane z konwersją. Przykładem może być JasperReports omawiany w dalszej części materiałów.

## Funkcje ogólne

Istnieją funkcje działające na wszystkich typach danych i odnoszą się do używania wartości *NULL* w obliczeniach. Należą do nich funkcje:

- *nvl* (expr1, expr2) - konwertuje *NULL* w expr1 na określoną wartość (expr2),
- *nullif* (expr1, expr2) - porównuje oba wyrażenia i zwraca *NULL*, gdy są one równe, w przeciwnym przypadku zwraca pierwsze wyrażenie,
- *coalesce* (expr1, expr2, ..., exprn) - zwraca pierwsze wyrażenie z listy, które nie jest *NULL*.

Poniżej zostanie omówiona tylko pierwsza z tych funkcji czyli *nvl*.

Jak już wcześniej zaznaczono - *NULL* ma tę właściwość, że działanie: wartość + *NULL* = *NULL*.

Wykonując zapytanie:

```
select nr_zawodnika, punkty_globalne, punkty_kategorii
from bd3_wyniki;
```

otrzymujemy zbiór, którego fragment jest przedstawiony poniżej:

NR_ZAWODNIKA	PUNKTY_GLOBALNE	PUNKTY_KATEGORII
849	2	35
850	(null)	(null)
851	(null)	(null)
853	(null)	18
855	(null)	10

Dla celów prezentacji działań na wartościach *NULL* rozszerzymy listę *select* o dodatkową kolumnę będącą sumą obu kolumn ze zdobytymi punktami<sup>4</sup>:

```
select nr_zawodnika, punkty_globalne, punkty_kategorii
, punkty_globalne + punkty_kategorii as suma_pkt
from bd3_wyniki;
```

Otrzymamy:

NR_ZAWODNIKA	PUNKTY_GLOBALNE	PUNKTY_KATEGORII	SUMA_PKT
849	2	35	37
850	(null)	(null)	(null)
851	(null)	(null)	(null)
853	(null)	18	(null)
855	(null)	10	(null)

Wyniki są błędne dla dwóch wierszy. Zastosowanie funkcji *nvl* pozwala je poprawić.

Powyższe zdanie można zbudować tak:

```
select nr_zawodnika, punkty_globalne, punkty_kategorii
, nvl ( punkty_globalne, 0 ) + nvl ( punkty_kategorii,0 ) as suma_pkt
from bd3_wyniki;
```

, a wyniki ulegną zmianie:

<sup>4</sup> W praktyce nie ma takiej klasyfikacji, w której należało dodawać te dwie wielkości. Należy traktować ten przykład jako demonstrację właściwości *NULL*.

NR_ZAWODNIKA	PUNKTY_GLOBALNE	PUNKTY_KATEGORII	SUMA_PKT
849	2	35	37
850	(null)	(null)	0
851	(null)	(null)	0
853	(null)	18	18
855	(null)	10	10

Argumentem funkcji *nvl* może być również kolumna tabeli typu *date* lub *varchar2*, na przykład:

```
.... nvl ( data_zapisu, sysdate ) .....
```

oznaczać będzie, że jeśli w kolumnie *data\_zapisu* pojawi się *NULL* to będzie on konwertowany na bieżącą datę. Podobnie dla wartości typu *varchar2*:

```
....nvl ( wartosc_towaru, 'Brak danych' ) .....
```

Uwaga:

Warto zastanawiać się nad koniecznością permanentnego stosowania funkcji *nvl* w przypadku, gdy kolumna numeryczna w tabeli podlegająca obliczeniom została zdefiniowana w zdaniu *create table...* jako mogąca przyjmować wartości *NULL*.

## Funkcje grupujące w zdaniach SQL

Język SQL umożliwia dokonywanie obliczeń na zbiorze wierszy tabeli, wynikiem których może być jedna wartość (skalar) lub kolumna. Operacje te są możliwe dzięki funkcjom grupującym:

- *count* - zliczanie wierszy ,
- *sum* - sumowanie wartości w kolumnie,
- *avg* - obliczanie średniej z wartości w kolumnie,
- *min* - znajdowanie minimalnej wartości w kolumnie,
- *max* - znajdowanie maksymalnej wartości w kolumnie.

**COUNT** – funkcja ta różni się od pozostałych funkcji grupujących, ponieważ zamiast prowadzić obliczenia na wartościach kolumn, zwraca tylko liczbę wierszy w tabeli.

Poniższe zapytanie zwraca liczbę zawodników płci męskiej z tabeli *BD3\_ZAWODNICY*:

```
select count ( * ) as "Mężczyźni"
from bd3_zawodnicy
where plec = 'M';      -- wynik: 641
```

Użycie znaku *\** jako argumentu funkcji spowoduje przeliczenie wszystkich wierszy w tabeli.

Jeśli zachodzi potrzeba wykluczenia z zapytania wierszy zawierających wartości *NULL* w jednej z kolumn, należy zamiast *\** wpisać nazwę tej wybranej kolumny:

```
select count ( nr_klubu ) as "Mężczyźni"
from bd3_zawodnicy
where plec = 'M';      -- wynik: 638
```

Co oznacza, że trzech mężczyzn nie należy do żadnego klubu.

Funkcji tej można także używać do określenia, ile różnych od siebie wartości pojawia się w interesującej nas kolumnie. Na przykład chcąc się dowiedzieć ile jest nienulowych wartości w kolumnie *Pozycja* w tabeli *BD3\_WYNIKI* należy użyć zdania:

```
select count ( pozycja ) -- zliczane są tylko wystąpienia nienulłowe
from bd3_wyniki
where nr_zawodow = 1;
```

natomiast zdanie:

```
select count ( distinct pozycja )
from bd3_wyniki
where nr_zawodow = 1;
```

zwróci liczbę różnych wartości występujących w kolumnie Pozycja.

```
Zdanie select count ( pozycja )          as "Ile no_null",
        count ( distinct pozycja )    as "Ile różnych no_null"
from bd3_wyniki
where nr_zawodow = 1;
```

wykona poniższe obliczenia:

Ile no_null	Ile różnych no_null
274	2

Co oznacza, że w tabeli BD3\_WYNIKI są 274 wyniki dla zawodów o numerze 1, ale tylko dwie różne wartości w kolumnie Pozycja.<sup>5</sup>

Przykłady użycia pozostałych funkcji:

```
select sum ( punkty_globalne ) as Suma_pkt_globalne
from bd3_wyniki w, bd3_zawodnicy z, bd3_kategorie k
where w.nr_zawodnika = z.nr_zawodnika
    and z.nr_kategorii = k.nr_kategorii
    and w.nr_zawodow=1
    and k.nazwa_kategorii = 'K-III' ;
```

```
select round ( avg ( 2021 - extract ( year from data_urodzenia )), 1) as "Średni_wiek"
from bd3_zawodnicy
where nr_klubu = 2;
```

```
select max ( punkty_kategorii ) as Max_pkt_335
    , min ( punkty_kategorii ) as Min_pkt_335
from bd3_wyniki
where nr_zawodnika = 335;
```

Należy zwrócić uwagę na działanie powyższych funkcji, w przypadku gdy w kolumnie będącej argumentem takiej funkcji występują wartości *NULL*.

Przy pomocy zapytania:

```
select nr_zawodnika, punkty_globalne
from bd3_wyniki
where nr_zawodnika = 934;
```

<sup>5</sup> Kolumna *Pozycja* w tabeli BD3\_WYNIKI symuluje działanie fotokomórki. Jeśli kilku zawodników osiągnęło na mecie ten sam czas to w kolumnie *Pozycja* określa się ich kolejność. Prezentowane wyniki każą wnioskować, że w tych zawodach zadziałała fotokomórka i w kolumnie *Pozycja* występują wartości: 1 i 2, a więc był co najmniej jeden przypadek, gdy dwóch zawodników przybiegło na metę w tym samym czasie, natomiast nie było przypadku, by trzech zawodników osiągnęło ten sam rezultat.

można się dowiedzieć jakie wyniki, w kontekście punktów globalnych, osiągnął konkretny zawodnik:

NR_ZAWODNIKA	PUNKTY_GLOBALNE
934	16
934	(null)
934	15

Zawodnik o numerze 934 startował trzy razy w sezonie, dwa razy zdobył punkty a raz nie.

Użycie dla powyższych danych funkcji agregujących przy pomocy zdania:

```
select max ( punkty_globalne ) as Max_pkt_934
      , min ( punkty_globalne ) as Min_pkt_934
      , sum ( punkty_globalne ) as Suma_pkt_934
      , avg ( punkty_globalne ) as Średnia_pkt_934
from bd3_wyniki
where nr_zawodnika = 934;
```

da poniższy zbiór wyników:

MAX_PKT_934	MIN_PKT_934	SUMA_PKT_934	ŚREDNIA_PKT_934
16	15	31	15,5

Uwagi:

1. Funkcje agregujące nie uwzględniają wartości *NULL* w realizacji obliczeń.
2. Interesujące są wyniki zwrócone przez funkcję *avg*. Zawodnik trzy razy brał udział w zawodach, ale średnia została policzona dla dwóch. Nie miejsce tutaj na roztrząsanie czy to jest prawidłowe czy nie. Trzeba postawić pytanie, czy można zastosować rozwiązanie uwzględniające ten trzeci bieg? Otóż można stosując funkcję *nvl*:

```
select avg ( punkty_globalne ) as Średnia_pkt_934_I
      , round ( avg ( nvl ( punkty_globalne , 0 ) ), 1) as Średnia_pkt_934_II
from bd3_wyniki
where nr_zawodnika = 934;
```

Otrzymamy inny wynik:

ŚREDNIA_PKT_934_I	ŚREDNIA_PKT_934_II
15,5	10,3

Który jest prawidłowy zależy od biznesowych założeń, ale widać, że język SQL jest przygotowany na obie ewentualności.

3. Można użyć do liczenia średniej konstrukcji *sum ( kolumna ) / count ( kolumna )* zamiast funkcji *avg*. Należy tylko pamiętać o zasadach przedstawionych powyżej, szczególnie gdy kolumna będąca argumentem funkcji może zawierać wartości *NULL*.

## Zapytania grupujące

Zapytania grupujące umożliwiają tworzenie macierzowych zbiorów wynikowych przy użyciu funkcji grupujących.

Aby posegregować dane w grupy, należy dodać do zdania *select* frazę *group by* zawierającą nazwy kolumn, których wartości mają zostać użyte przy formowaniu grup.

Na przykład zdanie:

```
select nr_klubu as "Nr klubu", count ( * ) as "Liczba zawodników"
from bd3_zawodnicy
group by nr_klubu
order by nr_klubu;
```

zwróci zestawienie obrazujące liczbę zawodników należących do poszczególnych klubów:

Nr klubu	Liczba zawodników
1	146
2	23
3	104
4	7
6	46
7	8
8	39
10	72
11	4

....

Należy bezwzględnie przestrzegać zasady, że w klauzuli *group by* muszą znaleźć się wszystkie kolumny poza funkcjami agregującymi, które występują na liście *select*.

W przypadku użycia zdania:

```
select nr_klubu as "Nr klubu", count(*) as "Liczba zawodników"
from bd3_zawodnicy
order by nr_klubu;
```

zostanie wygenerowany błąd:

```
Error report -
SQL Error: ORA-00937: to nie jest jednogrupowa funkcja grupowa
00937. 00000 - "not a single-group group function"
*Cause:
*Action:
```

gdyż brak jest frazy *group by* z kolumną *nr\_klubu*.

Chcąc tworzyć bardziej pogłębione raporty analityczne można stosować zagnieżdżanie grup stosując konsekwentnie zasadę opisaną powyżej.

Poniższe zdanie obrazuje grupowanie dwupoziomowe zliczające zawodników w klubach w podziale na kategorie wiekowe:

```
select nazwa_kategorii, nazwa_klubu, count ( * ) as "Liczba zawodników"
from bd3_zawodnicy z
      join bd3_kluby k on z.nr_klubu = k.nr_klubu
      join bd3_kategorie ka on z.nr_kategorii = ka.nr_kategorii
group by nazwa_klubu, nazwa_kategorii
order by nazwa_klubu, nazwa_kategorii;
```

Należy zwrócić uwagę na występowanie we frazie *select* obu kolumn, według których nastąpiło grupowanie.

NAZWA_KATEGORII	NAZWA_KLUBU	Liczba zawodników
VII	Bielański KB Warszawa	1
II	Gwardia Warszawa	1
III	Gwardia Warszawa	5
K-III	Gwardia Warszawa	1
K-V	Gwardia Warszawa	2
V	Gwardia Warszawa	1
VI	Gwardia Warszawa	2
III	KB Amator Mińsk Mazowiecki	1
IV	KB Amator Mińsk Mazowiecki	1
V	KB Amator Mińsk Mazowiecki	1
VIII	KB Amator Mińsk Mazowiecki	1
III	KB Gymnasion Warszawa	20
IV	KB Gymnasion Warszawa	20

....

Omówione do tej pory zapytania grupujące wykorzystują do utworzenia grup wszystkie wiersze w tabeli bazowej. Istnieją sposoby zmniejszenia liczby uwzględnianych wierszy:

- Ograniczenie liczby wierszy przed sformowaniem grup – przez użycie frazy *where*:

```
select nazwa_klubu, count ( * ) as "Liczba zawodników"
from bd3_zawodnicy z
      join bd3_kluby k on z.nr_klubu = k.nr_klubu
where nazwa_klubu like '%Warszawa%'
group by nazwa_klubu
order by nazwa_klubu;
```

- Utworzenie grup, a następnie ograniczenie ich liczebności – przez użycie frazy *having*:

```
select nazwa_klubu, count ( * ) as "Liczba zawodników"
from bd3_zawodnicy z
      join bd3_kluby k on z.nr_klubu = k.nr_klubu
group by nazwa_klubu
having count ( * ) > 30
order by "Liczba zawodników" desc;
```



NAZWA_KLUBU	Liczba zawodników
Allianz Warszawa	146
KB Gymnasion Warszawa	104
KB Trucht Warszawa	100
KB Lechici Zielonka	73
AZS Uniwersytet Warszawski	72
KB Promyk Ciechanów	46
AZS SGGW Warszawa	39
KB Legionowo	32
Warszawianka	32

Do zbioru wynikowego zostaną wstawione tylko te nazwy klubów wraz z ich licznosciami, w których liczność zawodników przekracza 30.

Uwagi:

1. We frazie *group by* oraz *having* nie można używać aliasów, natomiast w *order by* - tak.
2. Kolejność występowania kolumn we frazie *group by* nie ma znaczenia, natomiast we frazie *order by* - tak.
3. W przypadku użycia konstrukcji grupujących nie jest możliwe sortowanie według kolumny, która nie występuje we frazie *group by* (nie ma jej na liście *select*).

### Zadania do samodzielnego wykonania:

1. Opracować zestawienie zawodników zawierające imię i nazwisko jako jedną kolumnę o postaci A.ABACKI, wiek zawodnika i nazwę kategorii, dla zawodników, których numery zawarte są w przedziałach [30, 60] i [300,350]. Posortować zbiór malejąco według wieku zawodnika.
2. Obliczyć, ile kobiet należy do klubów o numerach z przedziału [10,20], które brały udział w zawodach nr 4.
3. Opracować zestawienie licznosci uczestników zawodów nr 3 o postaci:

*Nazwa kategorii, Liczba uczestników*

Posortować malejąco zbiór według licznosci uczestników.

4. Opracować zestawienie pokazujące sumaryczną liczbę zdobytych punktów w klasyfikacji generalnej ( *punkty\_globalne* ) przez poszczególne kluby w zawodach nr 1 uwzględniając tylko mężczyzn. Raport zawierający numery klubów posortować malejąco według zdobytych punktów i numerów klubów rosnąco i nie pokazywać w nim klubów, które nie zdobyły punktów.
5. Opracować zestawienie pokazujące sumaryczną liczbę zdobytych punktów w klasyfikacji generalnej przez poszczególne kluby w zawodach nr 3 uwzględniając tylko kobiety. Raport ten zawierający nazwy klubów posortować malejąco według zdobytych punktów i nie pokazywać w nim klubów, które zdobyły mniej niż 50 punktów.
6. Opracować zestawienie obrazujące średni wiek kobiet i mężczyzn uczestniczących w poszczególnych zawodach o postaci:

*Płeć, Data zawodów, Nazwa zawodów, Średni wiek, Liczba zawodników*

Średni wiek przedstawiać z dokładnością do jednego miejsca po przecinku (np. 42.4), a datę zawodów w formacie zawierającym pełną nazwę miesiąca. Zbiór wyników uporządkować według daty zawodów, płci oraz liczby zawodników malejąco.

7. Utworzyć tabelę mogącą przechowywać zagregowane dane dotyczące klubów o strukturze: nazwa klubu, liczności zarejestrowanych zawodników, średniej ich wieku. Wykorzystać konstrukcję *insert into table... select ....* .
8. Utworzyć tabelę mogącą przechowywać zagregowane dane dotyczące daty zawodów (o postaci YYYY-MM-DD) oraz liczby uczestniczących w nich zawodników. Wykorzystać konstrukcję *create table ...as select ...* .
9. Wykonać eksperyment polegający na uruchomieniu poniższych zdań zawierających funkcje agregujące:

```
select count ( * ) as "Liczba zawodników"
from bd3_zawodnicy;
```

```
select nr_klubu as "Nr klubu", count ( * ) as "Liczba zawodników"
from bd3_zawodnicy
group by nr_klubu;
```

```
select count ( * ) as "Liczba zawodników"
from bd3_zawodnicy
group by nr_klubu;
```

```
select avg (count ( * )) as "Średnia liczba zawodników"
from bd3_zawodnicy
group by nr_klubu;
```

```
select max (count ( * )) as "Maks liczba zawodników"
from bd3_zawodnicy
group by nr_klubu;
```

Zastanowić się nad interpretacją wyników, szczególnie trzech ostatnich zdań. Jest to o tyle ważne, że takie konstrukcje występują w bardziej skomplikowanych zdaniach SQL, na przykład w podzapytaniach.