

2.2.3 . Zapytania ze złączeniami relacji

Jeśli w ramach jednego zapytania pobierane są dane z więcej niż jednej relacji lub z różnych krotek tej samej relacji, należy te relacje (w drugim przypadku są to te same relacje) połączyć. Relacje mogą być łączone poziomo i pionowo. W przypadku połączenia poziomego relacje łączone wymieniane są po klauzurze FROM, w przypadku połączenia pionowego wynikowe relacje z dwu lub więcej zapytań łączone są jednym z operatorów zbiorowych (UNION, INTERSECT, MINUS).

Łączenie poziome

W przypadku połączenia poziomego krotka relacji wynikowej powstaje w wyniku konkatenacji krotek relacji łączonych (wymienionych w klauzurze FROM po przecinku lub w ramach operatora złączenia JOIN) spełniających tzw. warunek złączenia. Warunek złączenia musi zawierać w swojej definicji odniesienie do przynajmniej jednego atrybutu każdej z łączonych relacji. Jeśli w zapytaniu nie jest wykorzystywany operator złączenia JOIN (w klauzurze FROM nazwy relacji wymienione są po przecinku), realizowany jest iloczyn kartezjański łączonych relacji z ewentualną jego selekcją według warunku złączenia lub warunku selekcji wymienionych w klauzurze WHERE. W przypadku wykorzystania operatora złączenia, warunek złączenia występuje po klauzurze ON tego operatora (teta-złączenie). Najczęściej łączonymi relacjami są relacje powiązane ze sobą referencyjnie (warunek złączenia oparty jest na kluczach, głównym i obcym, relacji powiązanych).

Zad. Znaleźć kotki (płeć żeńska), które uczestniczyły w incydentach. Wyświetlić dodatkowo imiona wrogów uczestniczących w incydentach oraz opisy tych incydentów.

```
SQL> SELECT K.pseudo "Kotka", imie_wroga "jej wrog",  
2      opis_incydentu "Przewina wroga"  
3 FROM Kocury K, Wrogowie_kocurow WK  
4 WHERE K.pseudo=WK.pseudo AND plec='D';
```

Kotka	jej wrog	Przewina wroga
DAMA	KAZIO	CHCIAŁ OBEDRZEC ZE SKORY
KURKA	BUREK	POGONIL
LASKA	KAZIO	ZLAPAL ZA OGON I ZROBIL WIATRAK
LASKA	DZIKI BILL	POGRYZL ZE LEDWO SIE WYLIZALA
MALA	CHYTRUSEK	ZALECAL SIE
PUSZYSTA	SMUKLA	OBRZUCILA SZYSZKAMI
SZYBKĄ	GLUPIA ZOSKA	UZYLA KOTA JAKO SCIERKI
UCHO	SWAWOLNY DYZIO	OBRZUCIL KAMIENIAMI

8 wierszy zostało wybranych.

SQL>

Zapytanie powyższe realizuje znaną z algebry relacji operację równozłączenia z wykorzystaniem iloczynu kartezjańskiego łączonych relacji. Łączonym relacjom Kocury i Wrogowie_kocuw nadano aliasy, odpowiednio K i WK, aby jednoznacznie identyfikować atrybuty o tych samych nazwach pochodzące z obu relacji.

Zgodnie ze standardem SQL'a powyższe zadanie można równoważnie rozwiązać (dialekt SQL'a proponowany przez Oracle) z wykorzystaniem operatora teta-złączenia JOIN (tutaj równozłączenia), w standardzie ANSI SQL zapisywanego jako INNER JOIN (złączenie wewnętrzne – przeciwieństwo złączenia zewnętrznego; Oracle, ze względu na standard, akceptuje klauzulę INNER, choć jest ona nieobowiązkowa w jego składni), gdzie po klauzurze ON występuje warunek złączenia, do którego ewentualnie można także przenieść warunek selekcji umieszczony poniżej w klauzurze WHERE:

```
SQL> SELECT K.pseudo "Kotka", imie_wroga "jej wrog",  
2      opis_incydentu "Przewina wroga"  
3 FROM Kocury K JOIN Wrogowie_kocuw WK  
4      ON K.pseudo=WK.pseudo  
5 WHERE plec='D';
```

a także z wykorzystaniem zastępującej klauzulę ON klauzuli USING (równozłączenie ze względu na jeden lub więcej wspólnych, pod względem nazwy i typu, atrybutów; jedna ze wspólnych kolumn jest w relacji wynikowej pomijana):

```
SQL> SELECT pseudo "Kotka", imie_wroga "jej wrog",  
2      opis_incydentu "Przewina wroga"  
3 FROM Kocury JOIN Wrogowie_kocuw USING(pseudo)  
4 WHERE plec='D';
```

lub ewentualnie z wykorzystaniem operatora realizującego złączenie naturalne:

```
SQL> SELECT pseudo "Kotka", imie_wroga "jej wrog",  
2      opis_incydentu "Przewina wroga"  
3 FROM Kocury NATURAL JOIN Wrogowie_kocurow  
4 WHERE plec='D';
```

Należy tu zwrócić uwagę na fakt, że kolumny używane w złączeniu naturalnym i w złączeniu z użyciem klauzuli USING (kolumny, ze względu na które złączenie następuje) nie mogą być poprzedzane aliasem relacji, z której pochodzą. W powyższych dwóch zapytaniach w klauzurze SELECT nie mógłby się więc pojawić zapis K.pseudo, gdzie K jest aliasem relacji Kocury. Nie dotyczy to innych atrybutów łączonych relacji. Wynika to z tego, że, jak już wspomniano, jedno wystąpienie wspólnego atrybutu (tu pseudo) jest tu eliminowane.

Zad. Znaleźć koty polujące na polu posiadające wrogów o stopniu wrogości powyżej 5.

```
SQL> SELECT DISTINCT  
2      K.pseudo||' ' "Ma groznego wroga na polu"  
3 FROM Kocury K, Wrogowie_kocurow WK, Wrogowie W, Bandy B  
4 WHERE K.nr_bandy=B.nr_bandy AND K.pseudo=WK.pseudo AND  
5      WK.imie_wroga=W.imie_wroga AND  
6      teren IN ('POLE', 'CALOSC') AND stopien_wrogosci>5;
```

Ma groznego wroga na polu

BOLEK
LASKA
RURA
TYGRYS
SQL>

Powyższe zadanie można równoważnie rozwiązać z wykorzystaniem trzech operatorów realizujących teta-złączenie z ewentualnym przeniesieniem warunków selekcji do odpowiednich klauzul ON (do tych, w których atrybuty biorące udział w selekcji są już dostępne):

```
SQL> SELECT DISTINCT  
2      K.pseudo||' ' "Ma groznego wroga na polu"  
3 FROM Kocury K  
4      JOIN Wrogowie_kocurow WK ON K.pseudo=WK.pseudo  
5      JOIN Wrogowie W ON WK.imie_wroga=W.imie_wroga  
6      JOIN Bandy B ON K.nr_bandy=B.nr_bandy  
7 WHERE teren IN ('POLE', 'CALOSC')  
8      AND stopien_wrogosci>5;
```

a także z wykorzystaniem klauzuli USING lub np. z wykorzystaniem trzech operatorów realizujących złączenie naturalne:

```
SQL> SELECT DISTINCT
2         pseudo||' ' "Ma groznego wroga na polu"
3 FROM Kocury NATURAL JOIN Wrogowie_kocurow
4         NATURAL JOIN Wrogowie
5         NATURAL JOIN Bandy
6 WHERE teren IN ('POLE','CALOSC')
7         AND stopien_wrogosci>5;
```

Zad. W każdej z band, oprócz swojej, Tygrys umieścił szpiega. Można go rozpoznać po tym, że podlega on bezpośrednio Tygrysowi a nie szefowi bandy choć nie jest członkiem bandy Tygrysa. Znaleźć wszystkich szpiegów Tygrysa.

```
SQL> SELECT K1.pseudo "Szpieg",K1.nr_bandy "Banda"
2 FROM Kocury K1 JOIN Kocury K2 ON K1.szef=K2.pseudo AND
3                                K1.nr_bandy<>K2.nr_bandy
4 WHERE K1.szef='TYGRYS';
```

Szpieg	Banda
ZOMBI	3
LYSY	2
RAFA	4

SQL>

Powyższe zadanie zostało rozwiązane z wykorzystaniem operatora teta-złączenia łączącego relację Kocury z nią samą.

Zad. Znaleźć imiona kotów, które przystąpiły do stada wcześniej, niż ich bezpośredni przełożeni.

```
SQL> SELECT K1.imie||' ' "W stadzie przed szefem"
2 FROM Kocury K1,Kocury K2
3 WHERE K1.szef=K2.pseudo AND K1.w_stadku_od<K2.w_stadku_od
```

W stadzie przed szefem

ZUZIA

SQL>

Zadanie powyższe można równoważnie rozwiązać z wykorzystaniem np. teta-złączenia:

```
SQL> SELECT K1.imie||' ' "W stadzie przed szefem"  
2 FROM Kocury K1 JOIN Kocury K2 ON K1.szef=K2.pseudo AND  
3 K1.w_stadku_od<K2.w_stadku_od;
```

W powyższym rozwiązaniu nie można użyć narzucającego się teta-półzłączenia ponieważ nie jest ono w Oracle realizowane (realizowane jest półzłączenie naturalne – operator HALF NATURAL JOIN)

Zad. Wyświetlić imiona kotów, które do tej pory nie uczestniczyły w incydentach.

```
SQL> SELECT imie "Bez incydentu"  
2 FROM Kocury K LEFT JOIN Wrogowie_kocurow WK  
3 ON K.pseudo=WK.pseudo  
4 WHERE WK.pseudo IS NULL;
```

Bez incydentu

LUCEK

MICKA

PUCEK

SQL>

Do rozwiązania powyższego zadania został wykorzystany operator lewostronnego złączenia zewnętrznego LEFT JOIN. Do rozwiązania można wykorzystać także operator RIGHT JOIN, zmieniając jedynie kolejność łączonych relacji. Równoważnymi operatorami dla LEFT JOIN i RIGHT JOIN są, zgodnie ze standardem ANSI, operatory, odpowiednio LEFT OUTER JOIN i RIGHT OUTER JOIN. Oracle pozwala na pominięcie, jako nieznaczącego dla składni, słowa OUTER. Lewostronne i prawostronne złączenie naturalne działa tak samo jak złączenie wewnętrzne więc nie można go tu wykorzystać.

Zadanie powyższe można równoważnie rozwiązać wykorzystując charakterystyczny dla Oracle operator (+) umieszczany przy jednym z atrybutów warunku łączącego. Wykonywane jest wtedy lewostronne złączenie tej relacji, z której pochodzi atrybut nie opatrzony znakiem (+), z relacją, z której pochodzi atrybut opatrzony tym znakiem.

Od wersji Oracle 9i zalecane jest stosowanie, zamiast operatora (+), operatorów złączenia zewnętrznego (LEFT ..., RIGHT ..., FULL ...) ze względu na ich zgodność ze standardem ANSI języka SQL. Rozwiązanie powyższego zadania z wykorzystaniem operatora (+) ma postać:

```
SQL> SELECT imie "Bez incydentu"
2   FROM Kocury K,Wrogowie_kocurow WK
3   WHERE K.pseudo=WK.pseudo(+)
4          AND WK.pseudo IS NULL;
```

Operator (+) realizuje tutaj złączenie lewostronne relacji Kocury z relacją Wrogowie_kocurow.

Zad. Wyświetlić raport zwracający informacje o podwładnych i przełożonych kotów płci męskiej. Jeśli kot nie posiada podwładnego, należy to w raporcie zaznaczyć. Podobnie należy w raporcie zaznaczyć brak przełożonego.

```
SQL> SELECT NVL(K1.pseudo, 'Brak przełożonego') "Przełożony",
2           NVL(K2.pseudo, 'Brak podwładnego') "Podwładny"
3   FROM Kocury K1 FULL JOIN Kocury K2 ON K1.pseudo=K2.szef
4   WHERE DECODE(K1.pseudo, NULL, 'M', K1.plec)='M' AND
5          DECODE(K2.pseudo, NULL, 'M', K2.plec)='M'
6   ORDER BY 1;
```

Przełożony	Podwładny
BOLEK	Brak podwładnego
Brak przełożonego	TYGRYS
LYSY	PLACEK
LYSY	RURA
MALY	Brak podwładnego
MAN	Brak podwładnego
PLACEK	Brak podwładnego
RAFA	MAN
RAFA	MALY
RURA	Brak podwładnego
TYGRYS	BOLEK
TYGRYS	ZOMBI
TYGRYS	LYSY
TYGRYS	RAFA
ZERO	Brak podwładnego

15 wierszy zostało wybranych.

SQL>

W rozwiązaniu powyższego zadania wykorzystano operator pełnego złączenia zewnętrznego FULL JOIN (równoważnym jest tu, zgodny ze standardem ANSI, operator FULL OUTER JOIN). Funkcja NVL zwraca wartość pierwszego argumentu, jeśli jest on różny od NULL lub wartość drugiego argumentu jeśli pierwszy argument jest równy NULL.

Łączenie pionowe

W przypadku połączenia pionowego łączone relacje traktowane są jako zbiory krotek a relacja wynikowa jest rezultatem operacji zbiorowej na tych zbiorach. Aby doszło do połączenia, relacje składowe muszą posiadać tę samą liczbę atrybutów a ich typy muszą być odpowiednio takie same (relacje muszą posiadać ten sam schemat). Do łączenia pionowego w Oracle wykorzystywane są następujące operatory zbiorowe:

Nazwy relacji pochodzą pierwszej	UNION	- suma bez powtórzeń,	atrybutów wynikowej zawsze z łączonej
	UNION ALL	- suma z powtórzeniami,	
	INTERSECT	- iloczyn (przekrój),	
	MINUS	- różnica.	

relacji. W zapytaniu takim ORDER BY może wystąpić tylko raz jako jego ostatnia klauzula. Porządkowanie może się odbywać tylko ze względu na numery wyrażeń w klauzuli SELECT.

Zad. Określić, jakie funkcje pełnią koty w bandach nr 1 i nr 2.

```
SQL> SELECT funkcja FROM Kocury WHERE nr_bandy=1
      2  UNION
      3  SELECT funkcja FROM Kocury WHERE nr_bandy=2;
```

FUNKCJA

BANDZIOR

DZIELCZY

LAPACZ

LOWCZY

MILUSIA

SZEFUNIO

6 wierszy zostało wybranych.

SQL>

Zastosowanie operatora UNION zamiast UNION ALL pozwala na usunięcie powtarzających się krotek w relacji wynikowej (w tym przypadku funkcji pełnionych zarówno w bandzie nr 1 jak i 2).

2.2.4. Zapytania z podzapytaniem

Jak już wcześniej wspomniano, niektóre klauzule zapytania SELECT mogą zawierać zagnieżdżone klauzule SELECT, czyli tzw. podzapytania. W SQL'u Oracle'a dotyczy to klauzul SELECT, FROM, WHERE i HAVING, przy czym dla klauzuli SELECT podzapytanie musi zwracać tylko jedną wartość (relacja wynikowa musi się składać z jednej krotki i jednego atrybutu). W zależności od tego, czy podzapytanie odnosi się do zapytania zewnętrznego (poprzez atrybut z relacji z zapytania zewnętrznego poprzedzony aliasem tej relacji) czy też nie, wyróżnia się odpowiednio podzapytania skorelowane (związane) i nieskorelowane (niezwiązane). Podzapytanie skorelowane wykonywane jest dla każdej krotki zapytania zewnętrznego, podzapytanie nieskorelowane tylko raz na początku działania zapytania zewnętrznego. Podzapytanie nie może zawierać klauzuli ORDER BY (za wyjątkiem podzapytania w klauzuli

FROM). Podzapytanie skorelowane nie może występować w klauzurze FROM.

Poniższe zadania ilustrują działanie podzapytań (pierwsze dotyczą podzapytań nieskorelowanych a później także skorelowanych).

Zad. Znaleźć koty, które wykonują tę samą funkcję co LOLA.

```
SQL> SELECT imie "Zastepca LOLI",nr_bandy "jego banda"
2   FROM Kocury
3   WHERE funkcja=(SELECT funkcja
4                     FROM Kocury
5                     WHERE pseudo='LOLA')
6   AND pseudo!='LOLA';
```

Zastepca LOLI	jego banda
RUDA	1
BELA	2
SONIA	3

```
SQL>
```

Zadanie powyższe można rozwiązać w sposób równoważny stosując połączenie relacji Kocury z relacją Kocury:

```
SQL> SELECT K1.imie "Zastepca LOLI",
2         K1.nr_bandy "jego banda"
3   FROM Kocury K1,Kocury K2
4  WHERE K1.funkcja=K2.funkcja AND K2.pseudo='LOLA'
5        AND K1.pseudo!='LOLA';
```

Równoważnym rozwiązaniem do powyższego może też być rozwiązanie wykorzystujące operator teta-złączenia (równozłączenie).

Zad. Znaleźć koty, których przydział myszy jest większy od średniego przydziału w całym stadzie.

```
SQL> SELECT pseudo "Pseudonim",przydział_myszy "Zjada"
2   FROM Kocury
3  WHERE przydział_myszy>(SELECT AVG(NVL(przydział_myszy,0))
4                        FROM Kocury);
```

Pseudonim	Zjada
TYGRYS	103
LYSY	72
PLACEK	67
SZYBKA	65
RURA	56
ZOMBI	75
KURKA	61
RAFA	65

8 wierszy zostało wybranych.

SQL>

Zadanie powyższe można rozwiązać w sposób równoważny stosując połączenie relacji Kocury z relacją, wyznaczoną przez podzapytanie umieszczone w klauzurze FROM, zawierającą jedną liczbę określającą średnią wartość spożycia myszy w stadzie:

```
SQL> SELECT pseudo "Pseudonim", NVL(przydzial_myszy,0) "Zjada"
2   FROM Kocury, (SELECT AVG(NVL(przydzial_myszy,0)) sre
3                  FROM Kocury)
4  WHERE przydzial_myszy>sre;
```

lub stosując teta-złączenie relacji Kocury z relacją zwracaną przez podzapytanie:

```
SQL> SELECT pseudo "Pseudonim", NVL(przydzial_myszy,0) "Zjada"
2   FROM Kocury JOIN (SELECT AVG(NVL(przydzial_myszy,0)) sre
3                  FROM Kocury) ON przydzial_myszy>sre;
```

Zad. Znaleźć koty, których przydział myszy należy do listy najmniejszych przydziałów w poszczególnych bandach.

```
SQL> SELECT imie "Imie", NVL(przydzial_myszy,0) "Zjada",
2         nr_bandy "Banda"
3   FROM Kocury
4  WHERE NVL(przydzial_myszy,0) IN
5         (SELECT MIN(NVL(przydzial_myszy,0))
6          FROM Kocury
7          GROUP BY nr_bandy)
8  ORDER BY 3,2;
```

Imie	Zjada	Banda
-----	-----	-----
RUDA	22	1
BELA	24	2
SONIA	20	3
LATKA	40	4
DUDEK	40	4

SQL>

Rozwiązania zadania wykorzystujące złączenia relacji są analogiczne do rozwiązania poprzedniego zadania. W powyższym rozwiązaniu wykorzystano operator IN. W przypadku wykorzystywania w zapytaniu podzapytania z operatorem NOT IN Oracle zaleca zastąpienie tego operatora operatorem złączenia zewnętrznego lub operatorem NOT EXISTS (ten ostatni jest zdefiniowany w dalszej części wykładu). W obu tych przypadkach, w przeciwieństwie do operatora NOT IN, możliwe jest wykorzystanie przez system indeksów.

Zad. Znaleźć koty o najniższych przydziałach myszy w swoich bandach.

```
SQL> SELECT imie "Imie",przydzial_myszy "Zjada",
2          nr_bandy "Banda"
3 FROM Kocury
4 WHERE (NVL(przydzial_myszy,0),nr_bandy) IN
5        (SELECT MIN(NVL(przydzial_myszy,0)),
6          nr_bandy
7          FROM Kocury
8          GROUP BY nr_bandy)
9 ORDER BY nr_bandy;
```

Imie	Zjada	Banda
-----	-----	-----
RUDA	22	1
BELA	24	2
SONIA	20	3
LATKA	40	4
DUDEK	40	4

SQL>

Rozwiązanie równoważne wykorzystujące złączenie relacji jest następujące:

```
SQL> SELECT imie "Imie", NVL(przydzial_myszy,0) "Zjada",  
2         nr_bandy "Banda"  
3 FROM Kocury, (SELECT MIN(NVL(przydzial_myszy,0)) mi,  
4                 nr_bandy nb  
5                 FROM Kocury  
6                 GROUP BY nr_bandy)  
7 WHERE nr_bandy=nb AND NVL(przydzial_myszy,0)=mi  
8 ORDER BY nr_bandy;
```

Wykorzystanie dla powyższego zadania teta-złączenia relacji Kocury z relacją będącą wynikiem podzapytania daje następujący kod:

```
SQL> SELECT imie "Imie", NVL(przydzial_myszy,0) "Zjada",  
2         nr_bandy "Banda"  
3 FROM Kocury JOIN  
4     (SELECT MIN(NVL(przydzial_myszy,0)) mi, nr_bandy nb  
5     FROM Kocury  
6     GROUP BY nr_bandy) ON nr_bandy=nb AND  
7                             NVL(przydzial_myszy,0)=mi  
8 ORDER BY nr_bandy;
```

Kolejne rozwiązanie równoważne przy zastosowaniu tym razem podzapytania skorelowanego jest następujące:

```
SQL> SELECT imie "Imie", NVL(przydzial_myszy,0) "Zjada",  
2         nr_bandy "Banda"  
3 FROM Kocury K  
4 WHERE NVL(przydzial_myszy,0)=  
5         (SELECT MIN(NVL(przydzial_myszy,0))  
6         FROM Kocury  
7         WHERE nr_bandy=K.nr_bandy)  
8 ORDER BY nr_bandy;
```

Skorelowanie w powyższym zapytaniu następuje w warunku WHERE podzapytania poprzez atrybut nr_bandy z aliasem – atrybut bez aliasu pochodzi z relacji Kocury otwartej w podzapytaniu a atrybut z aliasem z relacji Kocury otwartej przez zapytanie zewnętrzne. Podzapytanie wykonywane jest dla każdej krotki zapytania zewnętrznego (dla każdego kota znajdowany jest minimalny przydział myszy wśród kotów należących do jego bandy).

W przypadku podzapytań umieszczonych w klauzurze WHERE, zwracających tylko jedną kolumnę, można stosować operator ANY lub operator ALL. Operatory te zawsze występują z operatorami relacji (=, <, >, <=, >=). Przykładowo wyrażenia:

a) $X > ANY$ podzapytanie,

b) $X < ALL$ podzapytanie,

przyjmą wartość TRUE jeśli X będzie większe od przynajmniej jednej wartości zwróconej przez podzapytanie (wyrażenie a)) lub jeśli X będzie mniejsze od każdej wartości zwróconej przez podzapytanie (wyrażenie b)). W podobny sposób można stosować operatory ANY i ALL dla pozostałych operatorów relacji.

Zad. Znaleźć koty, których przydział myszy jest większy od najniższego przydziału myszy w bandzie nr 4. W rozwiązaniu zadania wykorzystać operator ANY.

```
SQL> SELECT imie "Imie", NVL(przydzial_myszy,0) "Zjada",  
2         nr_bandy "Banda"  
3 FROM Kocury  
4 WHERE przydzial_myszy > ANY (SELECT DISTINCT  
5                               NVL(przydzial_myszy,0)  
6                               FROM Kocury  
7                               WHERE nr_bandy=4)  
8 ORDER BY "Zjada" DESC;
```

Imie	Zjada	Banda
MRUCZEK	103	1
KOREK	75	3
BOLEK	72	2
JACEK	67	2
PUCEK	65	4
ZUZIA	65	2
PUNIA	61	3
BARI	56	2
KSAWERY	51	4
MELA	51	4
CHYTRY	50	1
LUCEK	43	3

12 wierszy zostało wybranych.

SQL>

Jak już wcześniej wspomniano, podzapytania można także umieszczać w klauzuli HAVING.

Zad. Znaleźć bandy, w których średni przydział myszy jest wyższy od średniego przydziału w bandzie nr 3.

```
SQL> SELECT nr_bandy "Banda lepsza od bandy 3",
2          AVG(NVL(przydzial_myszy,0))
3          "Sredni przydzial w bandzie"
4 FROM Kocury
5 HAVING AVG(NVL(przydzial_myszy,0))>
6          (SELECT AVG(NVL(przydzial_myszy,0))
7          FROM Kocury
8          WHERE nr_bandy=3)
9 GROUP BY nr_bandy;
```

Banda lepsza od bandy 3	Sredni przydzial w bandzie
1	50
2	56,8

SQL>

Podzapytania można zagnieżdżać w sobie a także łączyć operatorami logicznymi (to ostatnie w klauzulach WHERE i HAVING).

Zad. Znaleźć koty, których przydział myszy jest wyższy od najwyższego przydziału w bandzie LACIACI MYSLIWI.

```
SQL> SELECT imie||'
2          "Lepszy od kazdego z LACIATYCH",
3          przydzial_myszy "Zjada"
4 FROM Kocury
5 WHERE przydzial_myszy>
6          (SELECT MAX(NVL(przydzial_myszy,0))
7          FROM Kocury
8          WHERE nr_bandy=
9          (SELECT nr_bandy
10         FROM Bandy
11         WHERE nazwa='LACIACI MYSLIWI'))
12 ORDER BY przydzial_myszy DESC;
```

Lepszy od kazdego z LACIATYCH	Zjada
MRUCZEK	103
KOREK	75
BOLEK	72
JACEK	67

SQL>

Zad. Znaleźć koty z band BIALI LOWCY i LACIACI MYSLIWI, których przydział myszy jest większy od średniej z całego stada.

```
SQL> SELECT pseudo||'          ' "Gora BIALI i LACIACI!"
  2   FROM Kocury
  3   WHERE nr_bandy IN (SELECT nr_bandy
  4                       FROM Bandy
  5                       WHERE nazwa IN ('BIALI LOWCY',
  6                                     'LACIACI MYSLIWI'))
  7   AND
  8   przydzial_myszy>
  9   (SELECT AVG(NVL(przydzial_myszy,0))
 10   FROM Kocury);
```

```
Gora BIALI i LACIACI!
-----
ZOMBI
KURKA
RAFA
```

SQL>

Podzapytanie można także umieszczać w klauzurze SELECT, pod warunkiem, że zwraca ono relację o jednym wierszu i jednej kolumnie (jedna wartość).

Zad. Dla każdego kota płci męskiej znaleźć średni przydział myszy w jego bandzie.

```
SQL> SELECT pseudo "Kot", (SELECT AVG(NVL(przydzial_myszy,0))
  2   FROM Kocury
  3   WHERE nr_bandy=K.nr_bandy)
  4   "Srednio w bandzie"
  5   FROM Kocury K
  6   WHERE plec='M';
```

Kot	Srednio w bandzie
-----	-----
TYGRYS	50
BOLEK	50
ZOMBI	49,75
LYSY	56,8
RAFA	49,4
MAN	49,4
PLACEK	56,8
RURA	56,8
ZERO	49,75
MALY	49,4

10 wierszy zostało wybranych.

SQL>

Podzapytanie wykorzystane w powyższym rozwiązaniu jest skorelowane.

W przypadku podzapytań skorelowanych często stosowany jest operator EXISTS (ew. NOT EXISTS). Operator ten służy do sprawdzenia, czy podzapytanie zwraca jakąkolwiek krotkę. Jeśli tak, przyjmuje on wartość TRUE, jeśli nie wartość FALSE.

Zad. Cała elita przywódcza kociego stada doszła do wniosku, że potencjalnych zagrożeniem dla ich władzy są koty, które wprowadzie nie posiadają podwładnych, jednak bywają zادیorne (posiadają wrogów) a jednocześnie ich przydział myszy jest co najmniej równy wartości $\min_myszy + (\max_myszy - \min_myszy) / 3$ (musieli się w przeszłości czymś zasłużyć), gdzie \min_myszy i \max_myszy jest określone ich funkcją. Znaleźć te koty.


```
SQL> SELECT pseudo||' ' "Do przeczołgania",
2      nazwa "Nazwa bandy"
3 FROM Kocury K,Bandy B
4 WHERE NOT EXISTS (SELECT pseudo
5                   FROM Kocury
6                   WHERE szef=K.pseudo)
7      AND K.nr_bandy=B.nr_bandy
8 INTERSECT
9 SELECT pseudo||' ',nazwa
10 FROM Kocury K,Bandy B, Funkcje F
11 WHERE K.nr_bandy=B.nr_bandy AND K.funkcja=F.funkcja
12      AND przydzial_myszy>
13      NVL(min_myszy,0)+
14      (NVL(max_myszy,0)-NVL(min_myszy,0))/3
15 INTERSECT
16 SELECT pseudo||' ',nazwa
17 FROM Kocury K,Bandy B
18 WHERE EXISTS (SELECT pseudo
19              FROM Wrogowie_kocurow
20              WHERE pseudo=K.pseudo)
21      AND K.nr_bandy=B.nr_bandy
22 ORDER BY 2,1;
```

Do przeczołgania Nazwa bandy

-----	-----
LASKA	CZARNI RYCERZE
PLACEK	CZARNI RYCERZE
RURA	CZARNI RYCERZE
SZYBKĄ	CZARNI RYCERZE
BOLEK	SZEFOSTWO

SQL>

Pierwsza z łączonych pionowo relacji (zwracana przez zapytanie) zawiera dane o kotach nie posiadających podwładnych, druga dane o kotach charakteryzujących się przydziałem myszy zdecydowanie wyższym od dolnej granicy „widełek” myszowych a trzecia dane o kotach posiadających wrogów. Iloczyn (część wspólna) tych relacji daje wynik.

2.2.5. Tabele przestawne

Mechanizm tabel przestawnych umożliwia przestawianie w relacji wynikowej zapytania SELECT kolumn w miejsce wierszy i

odwrotnie. Od wersji Oracle 11g mechanizm ten jest realizowany za pomocą klauzuli PIVOT umożliwiającej agregację wyniku i przedstawienie go w wierszach zamiast w kolumnach (rotacja). Wykorzystanie agregacji (funkcji grupowej) wskazuje, że PIVOT jest rodzajem operacji grupowej. Narzędzie to bardzo dobrze nadaje się do celów analitycznych, pozwalając wygodnie przeglądać np. tendencje. W zapytaniu SELECT klauzula PIVOT występuje w ramach klauzuli FROM po nazwie źródła danych (relacja wyspecyfikowana jawnie lub w postaci podzapytania lub wynikająca ze złączenia), które ma być rotowane tzn. realizowana jest składnia:

```
SELECT ...  
FROM źródło_danych  
    PIVOT  
    (  
        funkcja_agregująca  
        FOR kolumna_do_rotacji  
        IN zakres_danych_do_rotacji  
    )  
WHERE...
```

Klauzula FOR określa, z danych jakiej kolumny w źródle danych mają powstać kolumny (atrybuty) relacji powstałej po rotacji. Klauzula ta określa więc rotowaną kolumnę źródła danych. Jej wartości stają się nagłówkiem relacji po rotacji. Klauzula IN określa w postaci zbioru (elementy wymienione po przecinku), które kolumny relacji po rotacji mają być dostępne w relacji wynikowej. Według atrybutów wybieranych przez klauzulę SELECT ale nie występujących w klauzurze FOR i nie będących argumentem funkcji agregującej (nie dotyczy to funkcji COUNT(*) chyba, że zamiast * wystąpi wyrażenie) następuje grupowanie. Dla powstałych w ten sposób grup wykonywana jest funkcja agregująca.

Zad. Dla kotów z band BIALI LOWCY i CZARNI RYCERZE wyświetlić pełnioną przez nich funkcję i całkowity przydział myszy. W wyniku nie uwzględnąć funkcji SZEFUNIO.

```
SQL> SELECT funkcja, nazwa,
2         NVL(przydzial_myszy,0)+NVL(myszy_extra,0)
3     "Calkowity przydzial myszy"
4     FROM Kocury NATURAL JOIN BANDY
5     WHERE nazwa IN ('CZARNI RYCERZE', 'BIALI LOWCY') AND
6         funkcja != 'SZEFUNIO';
```

FUNKCJA	NAZWA	Calkowity przydzial myszy
BANDZIOR	CZARNI RYCERZE	93
LOWCZY	CZARNI RYCERZE	65
MILUSIA	CZARNI RYCERZE	52
LOWCZY	CZARNI RYCERZE	67
LAPACZ	CZARNI RYCERZE	56
BANDZIOR	BIALI LOWCY	88
LOWCZY	BIALI LOWCY	61
KOT	BIALI LOWCY	43
MILUSIA	BIALI LOWCY	55

Rozwiązanie powyższego zadania jest oczywiste. Zadanie można zmodyfikować do następującego problemu:

Zad. Dla band BIALI LOWCY i CZARNI RYCERZE wyświetli sumę całkowitych przydziałów myszy z podziałem na pełnione przez koty funkcje. W zestawieniu pominąć funkcję SZEFUNIO.

```
SQL> SELECT funkcja, nazwa,
2         SUM(NVL(przydzial_myszy,0)+NVL(myszy_extra,0))
3     "Suma przydzialow dla bandy"
4     FROM Kocury NATURAL JOIN BANDY
4     WHERE nazwa IN ('CZARNI RYCERZE', 'BIALI LOWCY')
5         AND funkcja != 'SZEFUNIO'
6     GROUP BY funkcja,nazwa;
```

FUNKCJA	NAZWA	Suma przydzialow dla bandy
BANDZIOR	CZARNI RYCERZE	93
MILUSIA	CZARNI RYCERZE	52
LAPACZ	CZARNI RYCERZE	56
LOWCZY	BIALI LOWCY	61
BANDZIOR	BIALI LOWCY	88
MILUSIA	BIALI LOWCY	55
KOT	BIALI LOWCY	43
LOWCZY	CZARNI RYCERZE	132

Powyższy wynik byłby dużo bardziej czytelny, gdyby przedstawić go w postaci tablicy, w której nazwy band byłyby nazwami kolejnych kolumn, w ramach których specyfikowana byłaby suma przydziałów myszy dla wybranych funkcji. Jest to zadanie, w którym można wykorzystać klauzulę PIVOT. Źródłem danych byłaby zdefiniowana przez podzapytanie relacja zwracająca funkcję kota, jego całkowity przydział myszy i nazwę bandy, do której należy. Funkcją agregującą PIVOT’a byłaby SUM na całkowitym przydziale myszy. Kolumną do rotacji byłaby nazwa bandy a zakresem rotowanych danych byłyby dwie wartości nazwy bandy: BIALI LOWCY i CZARNI RYCERZE. Zapis tak opisanego polecenia przedstawiony jest poniżej.

```
SQL> SELECT *
  2  2 FROM (SELECT funkcja, nazwa,
  3          NVL(przydzial_myszy,0)+NVL(myszy_extra,0)
  4          myszy_calk
  5          FROM Kocury NATURAL JOIN BANDY)
  6          PIVOT
  7          (
  8          SUM(myszy_calk)
  9          FOR nazwa
 10          IN ('CZARNI RYCERZE' "Banda CZARNI RYCERZE",
 11             'BIALI LOWCY' "Banda BIALI LOWCY")
 12          )
 13 WHERE funkcja != 'SZEFUNIO'
 14 ORDER BY "Banda CZARNI RYCERZE" DESC;
```

FUNKCJA	Banda CZARNI RYCERZE	Banda BIALI LOWCY
KOT		43
DZIELCZY		
LOWCZY	132	61
BANDZIOR	93	88
LAPACZ	56	
MILUSIA	52	55

Wymienione w zakresie danych PIVOT’a (operator IN) nazwy band posiadają zwoje aliasy. Oprócz nazw band (których wartości mają stanowić nagłówki kolumn) oraz całkowitego przydziału myszy (na który nakładana jest funkcja agregująca), źródło danych zwraca jedynie kolumnę funkcja. Ze względu na tę kolumnę następuje grupowanie. Gdyby zamiast SUM(myszy_calk) wystąpiłoby

COUNT(*), grupowanie wykonane by było także ze względu na myszy_calk. Jeśli źródło danych, oprócz funkcji, nazwy bandy i całkowitego przydziału myszy kota, zwracałoby jeszcze inne atrybuty, grupowanie zostałoby wykonane ze względu na funkcję i te atrybuty. Przykładowo, niech dodatkowo wybierana będzie płeć. Grupowanie następuje, jak pokazuje to poniższy wynik, ze względu na funkcję i płeć.

```
SQL> SELECT *
  2  2 FROM (SELECT funkcja, plec, nazwa,
  3          NVL(przydzial_myszy,0)+NVL(myszy_extra,0)
  4          myszy_calk
  5          FROM Kocury NATURAL JOIN BANDY)
  6          PIVOT
  7          (
  8          SUM(myszy_calk)
  9          FOR nazwa
 10          IN ('CZARNI RYCERZE' "Banda CZARNI RYCERZE",
 11             'BIALI LOWCY' "Banda BIALI LOWCY")
 12          )
 13 WHERE funkcja != 'SZEFUNIO'
 14 ORDER BY "Banda CZARNI RYCERZE" DESC;
```

FUNKCJA	P	Banda CZARNI RYCERZE	Banda BIALI LOWCY
BANDZIOR	M	93	88
DZIELCZY	M		
KOT	D		
KOT	M		43
LAPACZ	M	56	
LAPACZ	D		
LOWCZY	M	67	
LOWCZY	D	65	61
MILUSIA	D	52	55

2.2.6. Funkcje analityczne

Funkcje analityczne są konstrukcją języka SQL, w ramach których wykonywane są dodatkowe operacje grupowe na wierszach wynikowych zapytania tuż przed wykonaniem klauzuli ORDER BY. Dla każdego wiersza wynikowego zapytania określany jest zbiór wierszy, dla których ta operacja będzie wykonana. Zbiór ten nazywany jest **oknem wiersza** i jest on definiowany za pomocą tzw. **klauzuli analitycznej**. Określa ona rozmiar zbioru, którym może być liczba wierszy (klauzula ROWS) lub przedział wartości (klauzula RANGE). W ramach klauzuli analitycznej zbiór wszystkich wierszy wynikowych można najpierw podzielić na rozłączne grupy (klauzula PARTITION), a następnie w ramach grupy określić okno danego wiersza. Funkcje analityczne można jedynie stosować w ramach klauzuli SELECT i ORDER BY.

Jak łatwo zauważyć, część rozwiązań z wykorzystaniem funkcji analitycznych można zastąpić alternatywnie wykorzystaniem podzapytania skorelowanego (związanego) w klauzurze SELECT. W obu realizacjach funkcja grupowa wykonywana jest za każdym razem od nowa dla każdego wiersza wynikowego. Będzie to pokazane w przykładzie poniżej.

Składnia funkcji analitycznej jest następująca:

```
nazwa_funkcji([argument [, ...]])  
OVER ([[PARTITION BY wyrażenie [, ..]  
      [ORDER BY wyrażenie [, ...]  
      [{ROWS | RANGE} [BETWEEN  
      liczba1 PRECEDING  
      [AND liczba2 FOLLOWING]]]])
```

gdzie:

- Nazwa funkcji określa funkcję okna (jest to zwykle funkcja agregująca), jeśli wystąpi specyfikacja okna (klauszula ROWS lub RANGE). Jeśli specyfikacja okna nie wystąpi, nazwa ta określa funkcję rankingu (np. RANGE() lub DENSE_RANGE). Jeśli dodatkowo nie będzie ustalony porządek wierszy (brak klauzuli ORDER BY), nazwa funkcji określa funkcję raportującą (zwykle jest nią funkcja agregująca),
- PARTITION BY określa podział zbioru wynikowego wierszy na grupy. Jeśli klauzula ta zostanie pominięta, grupę stanowi zbiór wierszy wynikowych,
- ORDER BY określa porządek wierszy w ramach grupy określonej przez klauzulę PARTITION BY (jeśli ta wystąpi; jeśli nie, grupą jest zbiór wierszy wynikowych).
- ROWS specyfikuje okno poprzez określenie liczby wierszy (liczba1+liczba2+1, gdzie liczba1 określa liczbę wierszy przed wierszem bieżącym a liczba2 liczbę wierszy po bieżącym). W przypadku wystąpienia BETWEEN muszą wystąpić PREECEDING i FOLLOWING, w przeciwnym wypadku musi wystąpić tylko PREECEDING,
- RANGE specyfikuje okno poprzez określenie zakresu wierszy (od wiersza o pozycji o liczba1 mniejszej od pozycji wiersza bieżącego do wiersza o pozycji o liczba2 większej od pozycji wiersza bieżącego) W przypadku wystąpienia BETWEEN muszą wystąpić PREECEDING i FOLLOWING, w przeciwnym wypadku musi wystąpić tylko PREECEDING.

W przypadku pominięcia ROWS lub RANGE, okno danego wiersza pokrywa się z jego grupą.

Zad. Dla każdego kota określić sumaryczny przydział myszy oraz procentowy udział jego przydziału myszy w jego bandzie.

```
SQL> SELECT pseudo, NVL(przydzial_myszy,0),  
2      SUM(przydzial_myszy)  
3      OVER (PARTITION BY nr_bandy) Sum_w_bandzie,  
4      ROUND(100*NVL(przydzial_myszy,0)/  
5      (SUM(przydzial_myszy)  
6      OVER (PARTITION BY nr_bandy))) Proc_w_bandzie  
7 FROM Kocury  
8 ORDER BY nr_bandy;
```

PSEUDO	PRZYDZIAL_MYSZY	SUM_W_BANDZIE	PROC_W_BANDZIE
MALA	22	200	11
BOLEK	50	200	25
LOLA	25	200	13
TYGRYS	103	200	52
SZYBKA	65	284	23
LYSY	72	284	25
LASKA	24	284	8
RURA	56	284	20
PLACEK	67	284	24
ZOMBI	75	199	38
KURKA	61	199	31
ZERO	43	199	22
PUSZYSTA	20	199	10
RAFA	65	247	26
UCHO	40	247	16
MALY	40	247	16
DAMA	51	247	21
MAN	51	247	21

W powyższym przypadku okno obliczeniowe dla danego kota pokrywa się z grupą określoną przez PARTITION BY a funkcja SUM jest funkcją raportującą.

Powyższe zadanie można rozwiązać alternatywnie z wykorzystaniem podzapytania skorelowanego.


```
SQL> SELECT pseudo, NVL(przydzial_myszy,0),
2          (SELECT SUM(przydzial_myszy)
3            FROM Kocury
4            WHERE nr_bandy=K.nr_bandy) Sum_w_bandzie,
5          ROUND(100*NVL(przydzial_myszy,0)/
6                (SELECT SUM(przydzial_myszy)
7                  FROM Kocury
8                  WHERE nr_bandy=K.nr_bandy)) Proc_w_bandzie
9 FROM Kocury K
10 ORDER BY nr_bandy;
```

Zad. Dla każdego kota płci męskiej określić liczbę kotów tej samej płci o przydziale myszy co najwyżej o 5 mniejszym i o co najwyżej 10 większym od jego przydziału.

```
SQL> SELECT pseudo,
2          COUNT(*)
3          OVER (ORDER BY NVL(przydzial_myszy,0)
4                  RANGE BETWEEN 5 PRECEDING AND 10 FOLLOWING)
5          Liczba
6 FROM Kocury
7 WHERE plec='M';
```

PSEUDO	LICZBA
MALY	3
ZERO	4
BOLEK	3
MAN	3
RURA	3
RAFA	4
PLACEK	4
LYSY	3
ZOMBI	2
TYGRYS	1

Funkcją okna jest tym razem funkcja COUNT. Należy zauważyć, że co najmniej jeden kot zawsze spełnia warunek zadania – jest to kot, względem którego określono okno.

Zad. Dla każdego kota określonego przez nr jego bandy i datę jego przyjęcia do stada obliczyć średnią z trzech przydziałów myszy: jego przydziału, przydziału kota przyjętego do stada bezpośrednio przed tym kotem i przydziału kota przyjętego do stada bezpośrednio po nim.

```
SQL> SELECT pseudo, nr_bandy, w_stadku_od||' ',
2         ROUND(AVG(NVL(przydzial_myszy,0))
3         OVER (PARTITION BY nr_bandy ORDER BY w_stadku_od
4         ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING),1)
5         srednio
6 FROM Kocury;
```

PSEUDO	NR_BANDY	W_STADKU_OD	SREDNIO
TYGRYS	1	2002-01-01	76,5
BOLEK	1	2002-05-05	58,3
MALA	1	2006-09-17	32,3
LOLA	1	2009-10-14	23,5
SZYBKA	2	2006-07-21	68,5
LYSY	2	2006-08-15	53,7
LASKA	2	2008-02-01	54,3
PLACEK	2	2008-12-01	49
RURA	2	2009-09-01	61,5
ZOMBI	3	2004-03-16	68
KURKA	3	2008-01-01	59,7
ZERO	3	2010-03-01	41,3
PUSZYSTA	3	2010-11-18	31,5
RAFA	4	2006-10-15	58
MAN	4	2008-07-12	55,7
DAMA	4	2008-11-01	47,3
UCHO	4	2011-01-01	43,7
MALY	4	2011-05-15	40

Funkcją okna jest tu funkcja AVG. Zadanie w wersji tylko dla kotów przyjętych bezpośrednio przed przyjęciem danego kota miało by kształt:

```
SQL> SELECT pseudo, nr_bandy, w_stadku_od,
2         ROUND(AVG(NVL(przydzial_myszy,0))
3         OVER (PARTITION BY nr_bandy ORDER BY w_stadku_od
4         ROWS 1 PRECEDING),1)
5         srednio
6 FROM Kocury;
```

Oprócz funkcji okna i funkcji raportujących (w obu przypadkach są to zwykle funkcje agregujące) można, w omawianej składni funkcji analitycznych, wykorzystywać funkcje rankingu. Wyznaczają one pozycję danego wiersza w stosunku do innych wierszy partycji. Przykładami takich funkcji są:

- RANK() – określa pozycję wiersza w partycji w rankingu sportowym tzn. miejsca ex aequo powodują opuszczenie (brak) kolejnego miejsca/miejsc w numeracji.
- DENSE_RANK() – działa tak samo jak RANK(), tylko po miejscach ex aequo następuje kolejne miejsce w numeracji,
- ROW_NUMBER() – nadaje kolejne numery wierszom partycji,
- PERCENT_RANK() – określa w wartościach od 0 do 1 udział danego wiersza w stosunku do poprzedzających go wierszy partycji,
- CUME_DIST() – działa tak samo jak PERCENT_RANK(), tylko określany jest udział wiersza wobec poprzedzających go wierszy z nim łącznie.

Zad. Dla każdego poziomu hierarchii (level) wyświetlić koty o dwóch najdłuższych stażach.

```
SQL> SELECT poziom,pseudo,"W STADKU"  
2 FROM (SELECT pseudo,w_stadku_od "W STADKU", poziom,  
3          DENSE_RANK()  
4          OVER (PARTITION BY poziom  
5                ORDER BY w_stadku_od) pozycja  
6 FROM (SELECT level poziom,pseudo,w_stadku_od  
7        FROM Kocury  
8        CONNECT BY PRIOR pseudo=szef  
9        START WITH szef IS NULL))  
10 WHERE pozycja<3;
```

POZIOM	PSEUDO	W STADKU
1	TYGRYS	2002-01-01
2	BOLEK	2002-05-05
2	ZOMBI	2004-03-16
3	SZYBKA	2006-07-21
3	KURKA	2008-01-01
4	ZERO	2010-03-01

Zad. Przy założeniu, że wypłaty myszy dokonywane są zgodnie z kolejnością wyznaczoną przez datę wstąpienia do stada, określić pierwszą połowę wypłat.

```
SQL> SELECT pseudo, wypłata
2   FROM (SELECT pseudo,
3             NVL(przydział_myszy,0)+NVL(myszy_extra,0)
4             wypłata,
5             PERCENT_RANK() OVER (ORDER BY w_stadku_od) w
6   FROM Kocury)
7  WHERE w<=0.5;
```

PSEUDO	WYPLATA
TYGRYS	136
BOLEK	50
ZOMBI	88
SZYBKA	65
LYSY	93
MALA	64
RAFA	65
KURKA	61
LASKA	52

Oprócz przedstawionych powyżej rodzajów funkcji analitycznych (funkcje okna, funkcje raportujące, funkcje rankingu) można jeszcze wymienić funkcje LAG/LEAD, funkcje FIRST/LAST, odwrotne funkcje percentyli, funkcje rankingu hipotetycznego, funkcja WIDTH_BUCKET i funkcje statystyczne. Nie są one jednak przedmiotem zainteresowania niniejszego wykładu i nie będą tu przedstawiane.

2.2.7. Klauzula WITH

Poprzedzająca zapytanie SELECT klauzula WITH pozwala nadać nazwę relacjom powstałym w wyniku podzapytań umieszczonych w klauzurze FROM tego zapytania. Oracle optymalizując zapytanie implementuje takie podzapytanie w postaci widoku zmaterializowanego lub tabeli tymczasowej. Określone w klauzurze WITH nazwy można wykorzystać w zapytaniu, co sprawia, że staje się ono bardziej czytelne. Fragment składni zapytania SELECT z klauzulą WITH ma postać:

```
WITH {NazwaPodzapytania AS (podzapytanie) [, ...]}  
SELECT [DISTINCT | ALL] {wyrażenie [alias] [, ...]} | *  
FROM {NazwaRelacjiPerspektywyNazwaPodzapytania [alias]  
      [operator_złączenia  
      NazwaRelacjiPerspektywyNazwaPodzapytania [alias]  
      [ON warunek_złączenia]] [, ...]}  
Reszta_składni_polecenia_SELECT
```

Klauzulę WITH można stosować także w podzapytaniu i wykorzystywać ją w jego klauzurze FROM. Nazwy w niej zdefiniowane są widoczne w zapytaniu z nią związanym i we wszystkich jego podzapytaniach. Nazwy te mogą być też wykorzystywane w ramach klauzuli FROM definicji innych nazw, umieszczonych w klauzuli WITH.

Zad. Znaleźć koty, których przydział myszy jest większy od średniego przydziału w całym stadzie.

```
SQL> WITH Sr AS  
2      (SELECT AVG(NVL(przydzial_myszy,0)) sre  
3      FROM Kocury)  
4 SELECT pseudo "Pseudonim",NVL(przydzial_myszy,0) "Zjada"  
5 FROM Kocury JOIN Sr ON przydzial_myszy>sre;
```

Pseudonim	Zjada
-----	-----
TYGRYS	103
ZOMBI	75
LYSY	72
SZYBKA	65
RAFA	65
KURKA	61
PLACEK	67
RURA	56

Zad. Znaleźć koty płci żeńskiej, które uczestniczyły w incydentach z wrogami o stopniu wrogości powyżej 5.

```
SQL> WITH Panie AS
  2      (select pseudo
  3          FROM Kocury
  4          WHERE plec='D'),
  5      Incyenty5 AS
  6      (SELECT pseudo
  7          FROM Wrogowie_kocurow NATURAL JOIN Wrogowie
  8          WHERE stopien_wrogosci>5)
  9  SELECT DISTINCT pseudo "Zadziorne kotki"
 10  FROM Panie NATURAL JOIN Incyenty5;
```

Zadziorne kotki

UCHO
LASKA
DAMA

SQL>

lub alternatywnie

```
SQL> WITH Panie AS
  2      (select pseudo
  3          FROM Kocury
  4          WHERE plec='D'),
  5      Zadziorne_kotki AS
  6      (SELECT DISTINCT pseudo
  7          FROM Wrogowie_kocurow NATURAL JOIN Wrogowie
  8              NATURAL JOIN Panie
  9          WHERE stopien_wrogosci>5)
 10  SELECT pseudo "Zadziorne kotki"
 11  FROM Zadziorne_kotki;
```

2.3. Środowisko SQL*Plus

SQL*Plus jest prostym środowiskiem, w którym uruchamiane są zarówno polecenia SQL kończone znakiem średnika jak i polecenia SQL*Plus wprowadzane w jednej linii (brak średnika jako zakończenia). W przeciwieństwie do poleceń SQL, których zasięg ograniczony jest do jednego polecenia, polecenia SQL*Plus nie tracą zwykle ważności przez całą sesję. Polecenia obu języków mogą służyć do budowy skryptów generujących raporty. Środowisko SQL*Plus jest często zastępowane przez użytkowników dużo przyjaźniejszym środowiskiem oferowanym przez program SQL Developer. Nie wszystkie funkcjonalności SQL*Plus'a można w nim jednak zrealizować stąd SQL*Plus jest standardowo dołączany do każdego wydania systemu Oracle.

2.3.1. Parametryzacja zapytań za pomocą zmiennych

W Oracle w ramach środowiska SQL*Plus (także w aplikacji SQL Developer) zapytania SQL można parametryzować poprzez zmienne. Zmienne mogą mieć charakter lokalny jak i globalny.

Rodzaj zmiennej	Zasięg	Sposób definicji
Lokalna	Pojedyncze zapytanie	& przed zmienną
Globalna	Cała sesja lub do polecenia UNDEFINE	&& przed zmienną lub polecenie DEFINE lub polecenie ACCEPT

Zad. Wyświetlić imiona i przydziały myszy kotów z bandy określonej parametrycznie pełniące parametrycznie określoną funkcję.

```
SQL> SELECT imie,przydzial_myszy FROM Kocury  
2 WHERE nr_bandy = &banda AND funkcja = '&nazwa_funkcji';
```

Podaj wartość dla banda: 3

Podaj wartość dla nazwa_funkcji: MILUSIA

stare 2: WHERE nr_bandy = &banda AND funkcja = '&nazwa_funkcji'

nowe 2: WHERE nr_bandy = 3 AND funkcja = 'MILUSIA'

IMIE	PRZYDZIAŁ_MYSZY
SONIA	20

SQL>

Zad. Wyświetlić pseudonimy kotów pobrane z parametrycznie określonej relacji spełniających parametrycznie określony warunek. Wynik ma być uporządkowany zgodnie z wartością parametrycznie określonego wyrażenia. Dodatkowo parametrycznie zdefiniować wyrażenie, którego wartość ma być także wyświetlona.

SQL> SELECT pseudo,&co_liczysz "Rachunki"

2 FROM &tabela

3 WHERE &warunek

4 ORDER BY &porzadek_wg;

Proszę podać wartość dla co_liczysz:

ROUND(NVL(przydział_myszy,0)/31,2)

stare 1: SELECT pseudo,&co_liczysz "Rachunki"

nowe 1: SELECT pseudo,ROUND(NVL(przydział_myszy,0)/31,2)
"Rachunki"

Proszę podać wartość dla tabela: Kocury

stare 2: FROM &tabela

nowe 2: FROM Kocury

Proszę podać wartość dla warunek: myszy_extra IS NOT NULL

stare 3: WHERE &warunek

nowe 3: WHERE myszy_extra IS NOT NULL

Proszę podać wartość dla porzadek_wg: NVL(przydział_myszy,0)

stare 4: ORDER BY &porzadek_wg

nowe 4: ORDER BY NVL(przydział_myszy,0)

PSEUDO	Rachunki
--------	----------

-----	-----
-------	-------

PUSZYSTA	,65
----------	-----

MALA	,71
------	-----

LASKA	,77
-------	-----

LOLA	,81
------	-----

LYSY	2,32
------	------

ZOMBI	2,42
-------	------

TYGRYS	3,32
--------	------

7 wierszy zostało wybranych.

SQL>

Zad. Wyświetlić wartość parametrycznie wybranego atrybutu z relacji Bandy. Zapewnić możliwość wykorzystania tego atrybutu w następnych zapytaniach.

```
SQL> SELECT &atrybut  
       7 FROM Bandy;
```

```
Podaj wartość dla atrybut: nr_bandy  
stare   1: SELECT &atrybut  
nowe    1: SELECT nr_bandy
```

```
NR_BANDY  
-----  
         1  
         2  
         3  
         4  
         5
```

```
SQL>
```

Ze względu na to, że parametr atrybut ma charakter globalny, jego wartość może być wykorzystana w poniższym zapytaniu.

```
SQL> SELECT &atrybut  
       2 FROM Kocury  
       3 GROUP BY &atrybut;
```

```
stare   1: SELECT &atrybut  
nowe    1: SELECT nr_bandy  
stare   3: GROUP BY &atrybut  
nowe    3: GROUP BY nr_bandy
```

```
NR_BANDY  
-----  
         1  
         2  
         3  
         4
```

```
SQL>
```

Zad. Wyświetlić imiona, funkcje i roczne przydziały myszy wszystkich kotów. Roczny przydział myszy zdefiniować jako zmienna globalną.

```
SQL> DEFINE
      zjada='(NVL(przydzial_myszy,0)+NVL(myszy_extra,0))*12'
SQL> SELECT imie,funkcja,&zjada "Roczne spozycie myszy"
      2 FROM Kocury
      3 ORDER BY &zjada,&atrybut;
stare 1: SELECT imie,funkcja,&zjada "Roczne spozycie myszy"
nowe 1: SELECT imie,funkcja,
          (NVL(przydzial_myszy,0)+NVL(myszy_extra,0))*12
          "Roczne spozycie myszy"
stare 3: ORDER BY &zjada,&atrybut
nowe 3: ORDER BY
          (NVL(przydzial_myszy,0)+NVL(myszy_extra,0))*12,
          nr_bandy
```

IMIE	FUNKCJA	Roczne spozycie myszy
-----	-----	-----
LATKA	KOT	480
DUDEK	KOT	480
LUCEK	KOT	516
CHYTRY	DZIELCZY	600
MELA	LAPACZ	612
KSAWERY	LAPACZ	612
BELA	MILUSIA	624
SONIA	MILUSIA	660
BARI	LAPACZ	672
PUNIA	LOWCZY	732
RUDA	MILUSIA	768
ZUZIA	LOWCZY	780
PUCEK	LOWCZY	780
JACEK	LOWCZY	804
MICKA	MILUSIA	864
KOREK	BANDZIOR	1056
BOLEK	BANDZIOR	1116
MRUCZEK	SZEFUNIO	1632

18 wierszy zostało wybranych.

```
SQL> UNDEFINE atrybut
SQL> UNDEFINE zjada
SQL>
```

W rozwiązaniu zadania wykorzystana została globalna wartość zmiennej atrybut zdefiniowana w ramach poprzedniego zadania oraz globalna wartość zmiennej zjada zdefiniowana w ramach bieżącego zadania. Na koniec obie zmienne zostały usunięte z pamięci (polecenie UNDEFINE).

Konwersacyjne definiowanie zmiennej globalnej w plikach poleceń

Zmienna globalna może być definiowana w sposób konwersacyjny zgodnie ze składnią:

ACCEPT zmienna [NUMBER|CHAR]
[**NOPROMPT** | **PROMPT** 'tekst'] [**HIDE**]

gdzie po **PROMPT** określany jest wyświetlany tekst (**NOPROMPT** – brak tekstu) a użycie **HIDE** powoduje brak wyświetlenia na konsoli wpisywanych danych.

```
SQL> ACCEPT liczba PROMPT 'Podaj liczbę: ';  
Podaj liczbę: 5  
  
SQL> UNDEFINE liczba  
SQL>
```

Zapamiętywanie polecenia SQL w pliku i uruchamianie z pliku

Ostatnio wykonywane w środowisku SQL*Plus (otwartym jako administrator) zapytanie można zapisać w pliku stosując polecenie **SAVE** a następnie uruchomić je z pliku stosując polecenie **START**. Odbывается to według składni:

SAVE nazwa_pliku

START nazwa_pliku [{wyrażenie [...]}]

gdzie oddzielone spacjami wyrażenia (może ich być maksymalnie 9) określają aktualne wartości zmiennych dla zapisanego zapytania.

***Zad.** Zapisać do pliku polecenie wyświetlające określone parametrycznie dane pobrane z relacji Kocury a następnie uruchomić z pliku zapisane polecenie.*

```
SQL> SELECT &1,&2  
2 FROM Kocury  
3 WHERE funkcja = '&3';
```

itd

```
SQL> SAVE wybor
Utworzono file wybor.sql
SQL> START wybor imie NVL(przydzial_myszy,0)*12 LOWCZY
stare 1: SELECT &1,&2
nowe 1: SELECT imie,NVL(przydzial_myszy,0)*12
stare 3: WHERE funkcja = '&3'
nowe 3: WHERE funkcja = 'LOWCZY'
```

```
IMIE                      NVL(PRZYDZIAL_MYSZY,0)*12
-----
ZUZIA                      780
PUCEK                      780
PUNIA                      732
JACEK                      804
```

SQL>

2.3.2. Ustawienie parametrów środowiska SQL*Plus

Parametry określające cechy środowiska SQL*Plus definiowane są za pomocą polecenia SET o składni:

SET {parametr [...]}

Poniżej przedstawiono wybrane parametry.

Parametr	Opis
ECHO {OFF <u>ON</u> }	Określa wyświetlanie na ekranie linii pliku poleceń.
FEED[BACK] {liczba_krotek OFF <u>ON</u> }	Określa, czy będzie podawana liczba przetworzonych krotek (domyślnie liczba_krotek=6).
HEA[DING] {OFF <u>ON</u> }	Reguluje wyświetlanie nagłówka relacji wynikowej
LIN[ESIZE] liczba_znaków	Definiuje długość linii dla wyjściowych danych (domyślnie liczba_znaków=80).
LONG liczba_znaków	Definiuje liczbę znaków wyświetlanych dla typu LONG

	(domyślnie liczba_znaków=80).
NEWPAGE {liczba_linii NONE}	Określa liczbę linii między kolejnymi stronami raportu (domyślnie liczba_linii=1). Wartość 0 wymusza wysuw strony (wyjście na drukarkę) lub wyczyszczenie ekranu (wyjście na ekran).
NUMF[FORMAT] szablon	Określa format wyświetlania liczb. Dostępne szablony przedstawione są poniżej,
NUM[WIDTH] długość	Określa dopuszczalną szerokość dla danych numerycznych (domyślnie długość=10),
PAGES[SIZE] liczba_linii	Określa liczbę linii wyświetlanych (drukowanych) na stronie (domyślnie liczba_linii=24),
PAUSE {OFF ON tekst}	Określa zatrzymanie generowania danych wyjściowych po każdej stronie (oczekiwanie na ENTER) z ewentualnym komunikatem (tekst).
SPACE odstęp	Określa odstęp między kolumnami relacji wynikowej (domyślnie odstęp=1, maksymalna dopuszczalna wartość odstępu - 10),
TERM[OUT] {OFF ON}	Określa wyświetlanie poleceń zawartych w skryptach.
VERIFY {OFF ON}	Określa wyświetlanie linii polecenia zawierającej odwołanie do zmiennej przed i po podstawieniu.
WRAP {OFF ON}	Określa, czy linie przekraczające długość ustaloną parametrem LINESIZE mają być "zawijane"

	czy obcinane.
--	---------------

Poniżej przedstawiono dostępne szablony dla parametru NUMFORMAT.

Szablon	Opis
An	Format znakowy, n określa liczbę znaków.
9	Miejsce danej numerycznej. Przykładowo liczba 1234 w formacie 999999 jest przedstawiona jako ^v 1234.
0	wyświetlanie zer wiodących. Przykładowo liczba 1234 w formacie 099999 jest przedstawiona jak 001234.
\$	Ruchomy znak dolara. Przykładowo liczba 1234 w formacie \$999999 jest przedstawiona jako ^v \$1234.
.	Kropka dziesiętna. Przykładowo liczba 1234 w formacie 999999.99 jest przedstawiona jako ^v 1234.00.
,	Przecinek tysięcy. Przykładowo liczba 1234 w formacie 999,999 będzie przedstawiona jako ^v 1,234.
PR	Liczby ujemne w nawiasach, Przykładowo liczba -1234 w formacie 999999PR będzie przedstawiona jako ^v <1234>.
EEEE	Notacja wykładnicza. Przykładowo liczba 1234 w formacie 99.999EEEE będzie przedstawiona jako ^v 1.234E+03.
V	Mnożenie przez 10 ⁿ (n - liczba dziewiątek po V). Przykładowo liczba 1234 w formacie 9999V99 będzie przedstawiona jako 123400.

W ramach wyświetlanych wartości mogą pojawić się następujące błędy:

- wartość nie mieści się w formacie,
% - format niezgody z typem wartości.

Wartości aktualnych ustawień parametrów można wyświetlić za pomocą polecenia SHOW o składni:

SHOW {ALL | parametr}

Gdzie ALL oznacza wyświetlenie listy wszystkich aktualnych ustawień a parametr oznacza wyświetlenie ustawienia konkretnego parametru.

2.3.3. Specyfikacja sposobu wyświetlania kolumny

Sposób wyświetlania kolumny relacji wynikowej można określić poleceniem COLUMN zgodnie ze składnią:

COL[UMN] nazwa_kolumny | alias_kolumny [{opcja [...]}]

Alias kolumny zdefiniowany w poleceniu SELECT jest obowiązkowy w poleceniu COLUMN. Definicja kolumny obowiązuje przez całą sesję. Brak listy opcji powoduje wyświetlenie parametrów kolumny.

Poniżej przedstawiono wybrane opcje dla polecenia COLUMN.

Opcja	Opis
FORMAT szablon	Kolumna wyświetlana zgodnie z szablonem. Dostępne szablony są takie same jak dla parametru NUMFORMAT.
<u>WRAP</u> TRUNC WORD_WRAP	Określa akcje, gdy kolumna nie mieści się w zadanej szerokości (WRAP - "zawijanie", TRUNC - obcinanie, WORD_WRAP "zawijanie" całych słów).
CLEAR	Odwołanie zadanych parametrów kolumny.
HEADING 'tekst'	Określa tekst nagłówka kolumny (' ' w tekście - przejście w nagłówku do nowej linii).
JUSTIFY LEFT RIGHT	Definiuje justowanie (daty i łańcuchy standardowo justowane są lewostronnie, liczby prawostronnie).
LIKE nazwa_kolumny	Nakazuje kopiowanie formatu z podanej kolumny.
NEWLINE	Powoduje rozpoczęcie wyświetlania każdej wartości kolumny od nowej linii

	(także nagłówek kolumny).
NULL 'tekst'	Zamiast wartości NULL wyświetlany tekst.
<u>PRINT</u> NOPRINT	Wyświetlanie lub nie zawartości kolumny.
TEMP	Parametry kolumny określone tylko są na czas tylko jednego polecenia SELECT.

2.3.4. Tworzenie tytułów stron i treści stopek raportu

Do tworzenia tytułów stron i stopek raportów służą odpowiednio polecenia TTITLE i BTITLE o składni:

TTI[TLE] [{klauzula_specyfikacji [...]} | **OFF** | **ON**]

BTI[TLE] [{klauzula_specyfikacji [...]} | **OFF** | **ON**]

Rozkaz bez specyfikacji powoduje wyświetlenie aktualnej zawartości tytułu lub stopki. Specyfikacja składa się z klauzul oddzielonych spacjami. OFF i ON powodują odpowiednio „wyłączenie” i „włączenie” tytułu lub stopki.

Poniżej przedstawiono dostępne klauzule specyfikacji.

Klauzula specyfikacji	Opis
'tekst'	Określa cały lub częściowy tekst tytułu lub stopki.
COL n	Wyświetlanie na n-tej pozycji bieżącej linii.
SKIP n	Przejdź do nowej linii n razy (n=0 - powrót do początku bieżącej linii).
LEFT CENTER RIGHT	Dotyczy danych występujących po klauzuli lub danych do następnej klauzuli LEFT CENTER RIGHT (specyfikacja oparta na wartości LINESIZE).
FORMAT szablon	Dotyczy danych występujących po klauzuli do końca rozkazu lub danych do następnej klauzuli FORMAT. Szablony są

	takie same jak w poleceniu COLUMN.
TAB n	Powoduje skok o n punktów tabulacji (n ujemne - wstecz).

W definicji tytułów i stopek wykorzystuje się często następujące zmienne systemowe:

SQL.PNO	- nr bieżącej strony raportu,
SQL.LNO	- nr bieżącej linii na stronie,
SQL.USER	- nazwa użytkownika Oracle,
SQL.SQLCODE	- kod bieżącego błędu,
SQL.RELEASE	- nr wersji systemu Oracle.

Aby zdefiniowany raport działał poprawnie, należy go zapisać do pliku w postaci skryptu a następnie uruchomić skrypt w środowisku SQL*Plus za pomocą polecenia START.

Zad. Zdefiniować raport określający miesięczne i trzyletnie spożycie myszy przez każdego kota.

```
SET FEEDBACK OFF LINESIZE 62 PAUSE ON
SET PAUSE 'Kontynuacja - ENTER'
COLUMN nr_bandy FORMAT 09 HEADING 'Banda'
COLUMN funkcja FORMAT A9 HEADING 'Fucha' JUSTIFY CENTER
COLUMN pseudo FORMAT A10 HEADING 'Pseudonim|słuzbowy'
COLUMN przydzial_myszy FORMAT 999.99 -
      HEADING 'Zjada mies.|bez dodatkow'
COLUMN myszy_extra FORMAT 999.99 -
      HEADING 'Zjada mies.|dodatkowo' NULL 'Bez dodatku'
COLUMN zje FORMAT 99,999.99 HEADING 'Zje przez|trzy lata'
TTITLE 'PRZEROB MYSZY W STADKU|(Dane przygotowal Dzielczy)'
BTITLE SKIP CENTER 'Do wiadomosci Szefunia' SKIP 2
SELECT nr_bandy,funkcja,pseudo,przydzial_myszy,myszy_extra,
      (przydzial_myszy+NVL(myszy_extra,0))*36 zje
FROM Kocury;
SET FEEDBACK ON LINESIZE 80 PAUSE OFF
CLEAR COLUMNS
TTITLE OFF
BTITLE OFF
```

SQL> start C:\RAPORT

Kontynuacja - ENTER

Mon Mar 03

strona 1

PRZEROB MYSZY W STADKU
(Dane przygotował Dzielczy)

Banda	Fucha	Pseudonim sluzbowy	Zjada mies. bez dodatkow	Zjada mies. dodatkowo	Zje przez trzy lata
01	SZEFUNIO	TYGRYS	103.00	33.00	4,896.00
01	DZIELCZY	BOLEK	50.00	Bez dodatku	1,800.00
01	MILUSIA	LOLA	25.00	47.00	2,592.00
01	MILUSIA	MALA	22.00	42.00	2,304.00
02	BANDZIOR	LYSY	72.00	21.00	3,348.00
02	LOWCZY	PLACEK	67.00	Bez dodatku	2,412.00
02	LOWCZY	SZYBKA	65.00	Bez dodatku	2,340.00
02	LAPACZ	RURA	56.00	Bez dodatku	2,016.00
02	MILUSIA	LASKA	24.00	28.00	1,872.00
03	BANDZIOR	ZOMBI	75.00	13.00	3,168.00
03	LOWCZY	KURKA	61.00	Bez dodatku	2,196.00
03	MILUSIA	PUSZYSTA	20.00	35.00	1,980.00
03	KOT	ZERO	43.00	Bez dodatku	1,548.00

Do wiadomosci Szefunia

Kontynuacja - ENTER

Mon Mar 03

strona 2

PRZEROB MYSZY W STADKU
(Dane przygotował Dzielczy)

Banda	Fucha	Psudonim sluzbowy	Zjada mies. bez dodatkow	Zjada mies. dodatkowo	Zje przez trzy lata
04	LOWCZY	RAFA	65.00	Bez dodatku	2,340.00
04	LAPACZ	DAMA	51.00	Bez dodatku	1,836.00
04	LAPACZ	MAN	51.00	Bez dodatku	1,836.00
04	KOT	UCHO	40.00	Bez dodatku	1,440.00
04	KOT	MALY	40.00	Bez dodatku	1,440.00

Do wiadomosci Szefunia

Komenda COLUMN posiada dodatkową opcję o nazwie NEW_VALUE umożliwiającą przenoszenie wartości wybranych kolumn do zmiennych SQL.

```
SET FEEDBACK OFF
SET LINESIZE 62
COLUMN SYSDATE NEW_VALUE dzis NOPRINT
SELECT SYSDATE FROM Dual;
TTITLE LEFT 'Data: ' dzis RIGHT FORMAT 999 -
'Str: ' SQL.PNO SKIP LEFT 'Kot: ' SQL.USER -
SKIP CENTER 'Raport z data ' SKIP 2
TTITLE OFF
CLEAR COLUMNS
SET FEEDBACK ON
SET LINESIZE 80
```

```
SQL> START C:\RAD
```

```
Data: 2003-03-03
```

```
Str: 1
```

```
Kot: Z
```

```
Raport z data
```

Zad. Znaleźć funkcję, której pełnienie daje kotom średnio największy przydział myszy. W rozwiązaniu wykorzystać znaleziony wcześniej maksymalny średni przydział myszy dla wszystkich funkcji przekazując go do drugiego polecenia SQL za pośrednictwem komendy COLUMN.

```
SQL> COLUMN maxi NEW_VALUE maxi
SQL> SELECT MAX(AVG(przydzial_myszy)) maxi
2 FROM Kocury
3 GROUP BY funkcja;
```

```
MAXI
```

```
-----
103
```

1 wiersz został wybrany.

```
SQL> SELECT funkcja
1 FROM Kocury
2 GROUP BY funkcja
3 HAVING AVG(przydzial_myszy)=&maxi
stare 4: HAVING AVG(przydzial_myszy)=&maxi
nowe 4: HAVING AVG(przydzial_myszy)= 103
```

```
FUNKCJA
```

```
-----
SZEFUNIO
```

1 wiersz został wybrany.

```
SQL>
```

2.3.5. Podział raportu na sekcje

Raport można podzielić na sekcje przygotowując w ten sposób każdą sekcję lub cały raport do podliczeń, ewentualnie specyfikując dodatkowo dla uzyskanych sekcji rozkazy formatujące. Do podziału raportu na sekcje służy rozkaz BREAK o składni:

BREAK [{**ON** wyrażenie [{opcja [...]}] [...]]
[**ON REPORT**]

Łamanie raportu następuje przed kropką, w której zmienia się wartość wyrażenia (w szczególności wartość atrybutu) wyspecyfikowanego po BREAK. Logiczne jest więc wykorzystanie klauzuli porządkowania, ze względu na tę kolumnę lub wyrażenie, w poleceniu SELECT. Dane z każdej tak przygotowanej sekcji (także cały raport - jeśli wystąpi klauzula ON REPORT) mogą być podliczone (polecenie COMPUTE).

Poniżej przedstawiono opcje dla polecenia BREAK

Opcja	Opis
SKIP PAGE	Przejdźcie do nowej strony.
SKIP n	Przejdźcie do nowej linii n razy.
DUP[LICATE]	Wyświetlanie powtarzającej się w ramach sekcji wartości kolumny (domyślnie NODUP[LICATE]).

Definicje podziałów na sekcje pamiętane są w środowisku SQL*Plus, stąd aby uniknąć „niespodzianek” podczas wykonywania następnych zapytań należy te podziały usunąć poleceniem CLEAR BREAKS. Informację o istniejących podziałach można uzyskać za pomocą polecenia BREAK bez parametrów.

2.3.6. Podliczenia raportu

Podliczenia dla zdefiniowanych sekcji raportu lub całego raportu wykonywane są za pośrednictwem rozkazu COMPUTE o składni:

**COMP[UTE] [{funkcja [...]} OF {wyrażenie [...]}
ON {przerwanie [...]} [REPORT]]**

Przez przerwanie rozumie się wyrażenie wymienione po ON w klauzurze BREAK. Dla każdej sekcji wyznaczona zostanie wartość funkcji, której argumentem jest wyrażenie wymienione po OF.

Dostępne dla COMPUTE funkcje to: COU[NT], MAX[IMUM], MIN[IMUM], NUM[BER], STD, SUM, VAR[iance]

Nazwa funkcji może być po spacji poszerzona o klauzulę LABEL ‘tekst’, gdzie tekst jest niestandardowym opisem funkcji (np. Suma zamiast SUM).

Podobnie jak dla definicji złamań BREAK, definicje podliczeń należy usuwać. Wykorzystuje się do tego rozkaz CLEAR COMPUTES. Informację o istniejących definicjach podliczeń można uzyskać poleceniem COMPUTE bez parametrów

Zad. Zdefiniować raport określający miesięczne i trzyletnie spożycie myszy przez każdego kota. Dokonać podliczeń wyznaczając średni oraz sumaryczny przydział myszy w każdej bandzie oraz w całym stadzie.

```

SET FEEDBACK OFF LINESIZE 62 PAGESIZE 50
COLUMN nr_bandy FORMAT 09 HEADING 'Banda'
COLUMN funkcja FORMAT A9 HEADING 'Fucha' JUSTIFY CENTER
COLUMN pseudo FORMAT A10 HEADING 'Pseudonim|sluzbowy'
COLUMN przydzial_myszy FORMAT 999.99 -
      HEADING 'Zjada mies.|bez dodatkow'
COLUMN myszy_extra FORMAT 999.99 -
      HEADING 'Zjada mies.|dodatkowo' NULL '      0.00'
COLUMN zje FORMAT 99,999.99 HEADING 'Zje przez|trzy lata'
TTITLE 'PRZEROB MYSZY W STADKU|(Dane przygotowal Dzielczy)'
BTITLE SKIP CENTER 'Do wiadomosci Szefunia' SKIP 2
BREAK ON nr_bandy SKIP 1 ON funkcja ON REPORT
COMPUTE AVG LABEL 'Sred.' SUM label 'Razem' -
      OF przydzial_myszy myszy_extra ON nr_bandy REPORT
COMPUTE SUM LABEL 'Razem' OF zje ON nr_bandy
SELECT nr_bandy,funkcja,pseudo,przydzial_myszy,myszy_extra,
      (przydzial_myszy+NVL(myszy_extra,0))*36 zje
FROM Kocury
ORDER BY nr_bandy,funkcja;
SET LINESIZE 80 FEEDBACK ON PAGESIZE 24
CLEAR COLUMNS BREAKS COMPUTES
TTITLE OFF
BTITLE OFF

SQL> START C:\RAPORT1

```

Wto Lut 24

strona 1

PRZEROB MYSZY W STADKU
(Dane przygotowal Dzielczy)

Banda	Fucha	Pseudonim sluzbowy	Zjada mies. bez dodatkow	Zjada mies. dodatkowo	Zje przez trzy lata
01	DZIELCZY	BOLEK	50.00	0.00	1,800.00
	MILUSIA	LOLA	25.00	47.00	2,592.00
		MALA	22.00	42.00	2,304.00
	SZEFUNIO	TYGRYS	103.00	33.00	4,896.00

Sred.			50.00	40.67	
Razem			200.00	122.00	11,592.00
02	BANDZIOR	LYSY	72.00	21.00	3,348.00
	LAPACZ	RURA	56.00	0.00	2,016.00
	LOWCZY	PLACEK	67.00	0.00	2,412.00
		SZYBKA	65.00	0.00	2,340.00
	MILUSIA	LASKA	24.00	28.00	1,872.00

Sred.			56.80	24.50	
Razem			284.00	49.00	11,988.00

03	BANDZIOR	ZOMBI	75.00	13.00	3,168.00
	KOT	ZERO	43.00	0.00	1,548.00
	LOWCZY	KURKA	61.00	0.00	2,196.00
	MILUSIA	PUSZYSTA	20.00	35.00	1,980.00
*****	*****		-----	-----	-----
Sred.			49.75	24.00	
Razem			199.00	48.00	8,892.00
04	KOT	UCHO	40.00	0.00	1,440.00
		MALY	40.00	0.00	1,440.00
	LAPACZ	DAMA	51.00	0.00	1,836.00
		MAN	51.00	0.00	1,836.00
	LOWCZY	RAFA	65.00	0.00	2,340.00
*****	*****		-----	-----	-----
Sred.			49.40	0.00	
Razem			247.00	0.00	8,892.00
			-----	-----	
Sred.			51.67	31.29	
Razem			930.00	219.00	

Do wiadomosci Szefunia

2.3.7. Publikacja raportu w Internecie

Raporty tworzone w środowisku SQL*Plus można przeznaczać do publikacji w Internecie. Uzyskuje się to dzięki poleceniu SET MARKUP HTML ON, które powoduje konwersję raportu na format HTML.

Zad. Umożliwić publikację w Internecie raportu dotyczącego spożycia myszy w bandzie, której numer zadany jest parametrycznie.

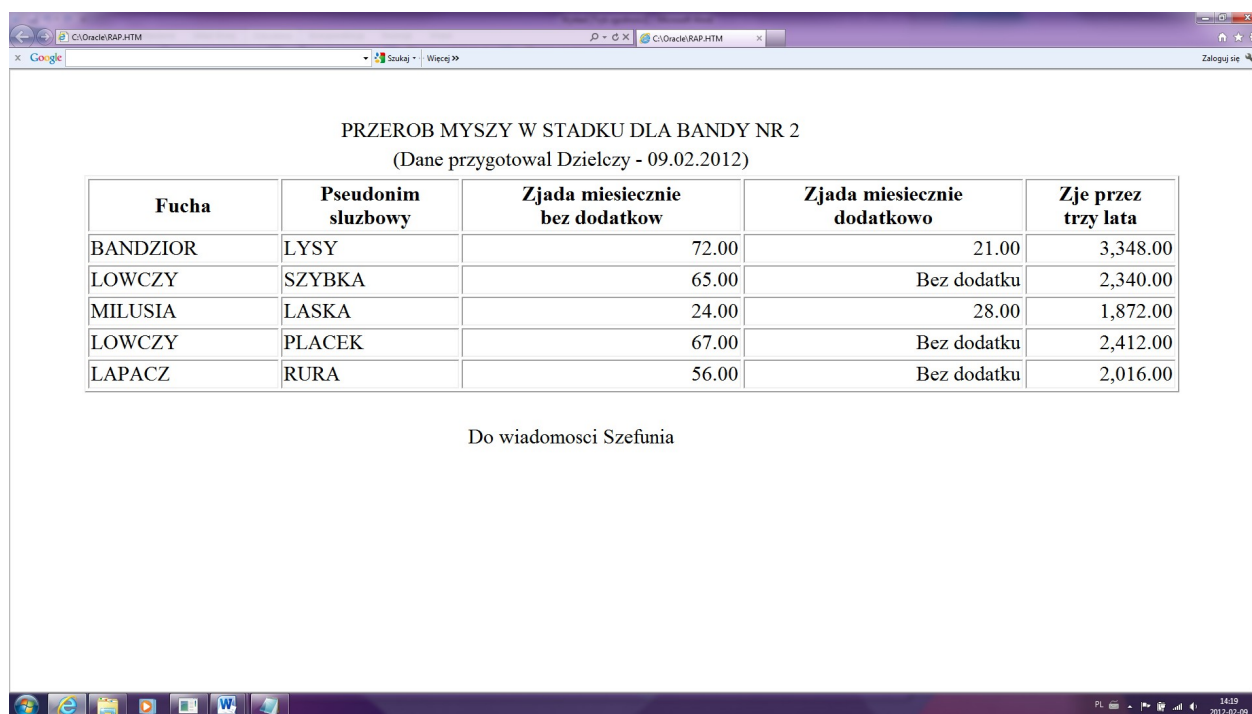
Poniżej przedstawiona jest zawartość skryptu RAP.SQL generującego raport. W skrypcie oprócz znanych już poleceń zastosowano rozkaz SPOOL przekierowujący do pliku RAP.HTM wyświetlany na ekranie raport.

```
ACCEPT nb PROMPT 'Podaj nr bandy: ';
SET FEEDBACK OFF TERMOUT OFF VERIFY OFF
SET MARKUP HTML ON
SET SPOOL ON
COLUMN TO_CHAR(SYSDATE,'DD.MM.YYYY') NEW_VALUE dzis NOPRINT
SELECT TO_CHAR(SYSDATE,'DD.MM.YYYY') FROM Dual;
COLUMN funkcja FORMAT A9 HEADING 'Fucha' JUSTIFY CENTER
COLUMN pseudo FORMAT A10 HEADING 'Pseudonim|słuzbowy'
COLUMN przydzial_myszy FORMAT 999.99 -
      HEADING 'Zjada miesiecznie|bez dodatkow'
COLUMN myszy_extra FORMAT 999.99 -
      HEADING 'Zjada miesiecznie|dodatkowo' NULL 'Bez dodatku'
COLUMN zje FORMAT 99,999.99 HEADING 'Zje przez|trzy lata'
TTITLE CENTER 'PRZEROB MYSZY W STADKU DLA BANDY NR ' nb -
      SKIP CENTER '(Dane przygotował Dzielczy - ' dzis '))'
BTITLE SKIP CENTER 'Do wiadomosci Szefunia'
SPOOL C:\ORACLE\RAP.HTM
SELECT funkcja,pseudo,przydzial_myszy,myszy_extra,
      (przydzial_myszy+NVL(myszy_extra,0))*36 zje
FROM Kocury
WHERE nr_bandy=&nb;
SET FEEDBACK ON TERMOUT ON VERIFY ON
SET MARKUP HTML OFF
CLEAR COLUMNS
TTITLE OFF
BTITLE OFF
SPOOL OFF
```

Skrypt uruchamiany jest w środowisku SQL*Plus poleceniem START.

```
SQL> START C:\ORACLE\RAP
Podaj nr bandy: 2
SQL>
```

Rezultatem działania skryptu RAP.SQL jest dokument HTML'owy o nazwie RAP.HTM, który wyświetlony na przeglądarce internetowej ma następującą postać:



PRZEROB MYSZY W STADKU DLA BANDY NR 2
(Dane przygotował Dziełczy - 09.02.2012)

Fucha	Pseudonim sluzbowy	Zjada miesiecznie bez dodatkow	Zjada miesiecznie dodatkowo	Zje przez trzy lata
BANDZIOR	LYSY	72.00	21.00	3,348.00
LOWCZY	SZYBKA	65.00	Bez dodatku	2,340.00
MILUSIA	LASKA	24.00	28.00	1,872.00
LOWCZY	PLACEK	67.00	Bez dodatku	2,412.00
LAPACZ	RURA	56.00	Bez dodatku	2,016.00

Do wiadomosci Szefunia

Raport w środowisku SQL*Plus (pominięcie poleceń SET TERMOUT ON, SET MARKUP HTML ON i SPOOL) będzie miał następujący kształt:

```
SQL> START C:\ORACLE\RAP1
Podaj nr bandy: 2
```

```

      PRZEROB MYSZY W STADKU DLA BANDY NR 2
(Dane przygotował Dziełczy - 09.02.2012)

      Fucha      Pseudonim  Zjada mies. Zjada mies.  Zje przez
      Fucha      sluzbowy   bez dodatkow dodatkowo   trzy lata
-----
BANDZIOR  LYSY           72.00       21.00    3,348.00
LOWCZY    PLACEK        67.00    Bez dodatku  2,412.00
LOWCZY    SZYBKA        65.00    Bez dodatku  2,340.00
LAPACZ    RURA         56.00    Bez dodatku  2,016.00
MILUSIA   LASKA         24.00       28.00    1,872.00
```

Do wiadomosci Szefunia

```
SQL>
```

2.4. Modyfikacja danych w relacji

Do specyfikacji poleceń modyfikujących dane w relacji służy składowa DML (ang. Data Manipulation Language) języka SQL. Można w jej ramach wyróżnić polecenie INSERT dodające nowe krotki do relacji, polecenie UPDATE modyfikujące istniejące w relacji dane, polecenie DELETE usuwające z relacji krotki i polecenie MERGE aktualizujące relacje z jednoczesnym wykonaniem operacji INSERT i UPDATE.

2.4.1. Polecenie INSERT

Polecenie INSERT służy do wstawienia jednego lub więcej wierszy bezpośrednio lub pośrednio do istniejącej relacji. Ten drugi przypadek (wstawianie pośrednie) zachodzi gdy wstawianie odbywa się poprzez perspektywę prostą nazywaną inaczej perspektywą modyfikowalną (oba te pojęcia zostaną przedstawione w dalszej części wykładu). Składnia polecenia INSERT jest następująca:

INSERT INTO NazwaRelacjiPerspektywy [({ atrybut [, ...] })]
{ **VALUES** ({ wartość [, ...] }) } | podzapytanie

Lista wymienionych po przecinku atrybutów określa nazwy atrybutów, których wartości będą wypełniane. Wszystkie nie wymienione w liście atrybuty muszą być nieobowiązkowe (NULL) lub posiadać zdefiniowaną wartość domyślną (określoną w poleceniu CREATE TABLE - polecenie składowej DDL języka SQL). Brak listy atrybutów w poleceniu wskazuje, że będą wypełniane wszystkie atrybuty relacji w kolejności ich definicji w poleceniu CREATE TABLE. Dane mogą być specyfikowane jawnie w klauzurze VALUES poprzez listę wymienionych po przecinku wartości lub niejawnie za pośrednictwem przygotowującego dane podzapytania. W pierwszym przypadku wstawiana jest do relacji jedna krotka, w drugim tyle krotek ile krotek zwraca podzapytanie. Liczba wartości wpisywana jawnie jak i liczba wartości zwracana przez podzapytanie musi być równa liczbie atrybutów wyspecyfikowanych w liście

atrybutów (jeśli ta wystąpi) a typy tych wartości muszą być zgodne z typami odpowiednich atrybutów. Istnieje też wersja polecenia INSERT pozwalająca na wstawianie wielu krotek w ramach jednego takiego polecenia. Wersja skrócona tego polecenia posiada następującą składnię:

INSERT ALL

```
{INTO NazwaRelacjiPerspektywy [( {atrybut [, ...]} )]  
  VALUES ( {wartość [, ...]} ) [ ... ]  
 {SELECT * FROM Dual} | podzapytanie
```

Powyższa wersja polecenia INSERT skraca czas ładowania danych do bazy danych (tylko jedno połączenie z bazą) i może być stosowana do wsadowego przepisywania danych z jednej bazy do drugiej, kiedy to istnieje pewność, że dane źródłowe są poprawne. W wersji tej możliwe jest także wprowadzanie, w ramach jednego polecenia, krotek do wielu różnych relacji. Źródłem danych może też podzapytanie zwracające krotki do wstawienia. W tym przypadku wartościami w klauzuli VALUES będą nazwy wyrażeń (ich aliasy) lub nazwy atrybutów zwracanych przez podzapytanie.

Zad. Tygrys postanowił karać niesubordynację swoich podwładnych poprzez czasowe zsyłanie ich do piwnic należących do mieszkańców wsi Wólka Mała. Zdefiniować w ramach relacji Bandy nową bandę o nazwie *Nieszczesnicy*, których terenem polowań będą piwnice.

```
SQL> INSERT INTO Bandy (nr_bandy,nazwa,teren)  
      2  VALUES(6,'NIESZCZESNICY','PIWNICE');
```

1 wiersz został utworzony.

```
SQL> ROLLBACK;
```

Wycofywanie zostało zakończone.

```
SQL>
```

Wartości atrybutów po klauzurze VALUES są wymienione zgodnie z kolejnością ich definicji w polecenie CREATE TABLE, można więc pominąć nazwy atrybutów i zapisać równoważnie powyższe polecenie w sposób następujący:

```
SQL> INSERT INTO Bandy
      2  VALUES(6, 'NIESZCZESNICY', 'PIWNICE');
```

1 wiersz został utworzony.

```
SQL> ROLLBACK;
```

Wycofywanie zostało zakończone.

```
SQL>
```

W ramach środowiska SQL*Plus polecenie INSERT można parametryzować. Powyższe polecenie mogłoby więc mieć kształt:

```
SQL> INSERT INTO Bandy
      2  VALUES (&id_bandy,&nazwa_bandy,&zasieg);
Proszę podać wartość dla id_bandy: 6
Proszę podać wartość dla nazwa_bandy: 'NIESZCZESNICY'
Proszę podać wartość dla zasieg: 'PIWNICE'
stare   2: VALUES (&id_bandy,&nazwa_bandy,&zasieg)
nowe    2: VALUES (6, 'NIESZCZESNICY', 'PIWNICE')
```

1 wiersz został utworzony.

```
SQL> ROLLBACK;
```

Wycofywanie zostało zakończone.

```
SQL>
```

Zad. Kilka kotów, do tej pory niezrzeszonych, postanowiło przyłączyć się do stada. Aby ułatwić swoje przyjęcie, przygotowali relację o nazwie *Nowi* z atrybutami *ksywa*, *imie*, *sex*, w której zapisali swoje dane. Wykorzystując relację *Nowi* rozszerzyć stado o nowych członków.

```
SQL> INSERT INTO Kocury
      2  (pseudo, imie, plec, w_stadku_od, szef)
      3  SELECT ksywa, imie, sex, SYSDATE, 'TYGRYS'
      4  FROM Nowi;
```

5 wierszy zostało utworzonych.

```
SQL> ROLLBACK;
```

Wycofywanie zostało zakończone.

```
SQL>
```

Atrybuty relacji Kocury, które nie uzyskują wartości w ramach powyższego polecenia są nieobowiązkowe.

2.4.2. Polecenie UPDATE

Polecenie UPDATE służy do zmiany wartości wybranych atrybutów w wybranych krotkach bezpośrednio we wskazanej relacji lub pośrednio poprzez perspektywę prostą nazywaną inaczej perspektywą modyfikowalną (ostatnie pojęcie zostanie przedstawione w dalszej części wykładu). Składnia polecenia jest następująca:

UPDATE NazwaRelacjiPerspektywy [alias]
SET {nazwa_atrybutu = wyrażenie [, ...]}
[**WHERE** warunek]

Po klauzurze UPDATE określana jest nazwa modyfikowanej relacji (ew. perspektywy modyfikowalnej). Alias oznacza nazwę alternatywną (zastępczą). Modyfikowane są wartości atrybutów wymienionych po klauzurze SET. Atrybuty te uzyskują nowe wartości określone przez wyrażenie (jego szczególnym przypadkiem jest stała). W ramach wyrażenia definiującego nową wartość atrybutu może wystąpić podzapytanie. O tym, w których krotkach następuje modyfikacja decyduje warunek po klauzurze WHERE polecenia. Warunek ten może być także określony przez podzapytanie. Brak klauzuli WHERE w poleceniu UPDATE powoduje modyfikację wartości wybranych atrybutów we wszystkich krotkach relacji.

Zad. Awansować kotkę o imieniu LATKA do funkcji LAPACZ dając jej jednocześnie 30 procentową „podwyżkę”

```
SQL> UPDATE Kocury
  2  SET funkcja='LAPACZ',
  3      przydzial_myszy=ROUND(NVL(przydzial_myszy,0)*1.3,0)
  4  WHERE imie='LATKA';
```

1 wiersz został zmodyfikowany.

```
SQL> ROLLBACK;
```

Wycofywanie zostało zakończone.

```
SQL>
```

Zad. Tygrys, jako światły władca postanowił, że nie będzie gospodarował dodatkowymi przydziałami myszy pod wpływem bieżących emocji. Zamiast tego zdecydował się na co miesięczne gratyfikacje określone na podstawie zapamiętywanych przez miesiąc w relacji Dodatki_extra „zasług” kotów. Zmodyfikować wartości przydziałów dodatkowych myszy w oparciu o „zasługi” kotów zapamiętane w relacji Dodatki_extra.

Przykładowa zawartość relacji Dodatki_extra:

```
SQL> SELECT pseudo "Pseudonim", dod_extra "Dodatek ekstra"  
2 FROM Dodatki_extra;
```

Pseudonim	Dodatek ekstra
TYGRYS	10
LOLA	5
DZIELCZY	10
TYGRYS	5
LOLA	-2

```
SQL> UPDATE Kocury K  
2 SET myszy_extra=  
3 (SELECT NVL(K.myszy_extra,0)+SUM(dod_extra)  
4 FROM Dodatki_extra D  
5 WHERE D.pseudo = K.pseudo)  
6 WHERE pseudo IN (SELECT pseudo  
FROM Dodatki_extra);
```

3 wiersze zostały zmodyfikowane.

```
SQL> ROLLBACK;
```

Wycofywanie zostało zakończone.

```
SQL>
```

W powyższym poleceniu wykorzystano podzapytanie skorelowane do modyfikacji wartości przydziałów dodatkowych i podzapytanie nieskorelowane do określenia kotów, których dane podlegają modyfikacji.

Podczas modyfikacji relacji można nadać atrybutowi wartość NULL. Jest to jedyny przypadek gdy w obsłudze wartości NULL jest wykorzystywany operator = zamiast operatora IS.

Zad. *W ramach walki z kryzysem Tygrys zarządził czasowe zawieszenie wydawania dodatkowych przydziałów myszy. Zrealizować to zadanie poprzez odpowiednią modyfikację atrybutu myszy_extra w relacji Kocury.*

```
SQL> UPDATE Kocury  
2 SET myszy_extra=NULL  
3 WHERE pseudo<>'TYGRYS';
```

17 wierszy zostało zmodyfikowanych.

```
SQL> ROLLBACK;
```

Wycofywanie zostało zakończone.

```
SQL>
```


2.4.3. Polecenie DELETE

Polecenie DELETE służy do usuwania wybranych krotek bezpośrednio ze wskazanej relacji lub pośrednio poprzez perspektywę prostą nazywaną inaczej perspektywą modyfikowalną (ostatnie pojęcie zostanie przedstawione w dalszej części wykładu). Składnia polecenia jest następująca:

DELETE FROM NazwaRelacjiPerspektywy
[**WHERE** warunek]

Po klauzurze DELETE FROM określana jest nazwa modyfikowanej relacji (ew. perspektywy modyfikowalnej). O tym, które krotki będą usuwane decyduje warunek po klauzurze WHERE. Jeśli klauzula ta nie wystąpi, polecenie DELETE powoduje usunięcie wszystkich krotek relacji.

***Zad.** Jak się jednak okazuje, bycie światłym władcą ma też swoje granice. Przedstawić wysoce zrozumiałą reakcję Tygrysa (usunięcie swoich danych z relacji Dodatki_extra) na zapowiedź wizyty Kociej Izby Kontroli.*

```
SQL> DELETE FROM Dodatki_extra  
2 WHERE pseudo='TYGRYS';
```

2 wiersze zostały usunięte.

```
SQL>
```

2.4.4. Polecenie MERGE

Polecenie MERGE pozwala na aktualizację relacji z jednoczesnym wykonaniem operacji INSERT i UPDATE. Dane do modyfikacji są pobierane z innej tabeli lub perspektywy i w zależności od spełnienia zawartych w poleceniu MERGE warunków modyfikują relację docelową.

Zad. Tygrys wysłał w teren swoich przedstawicieli z zadaniem werbowania nowych członków stada. Dane przedstawicieli zostały zapisane w relacji o nazwie *Ekspedycja* z atrybutami *ksywa*, *imie* i *sex*. Przedstawiciele mieli za zadanie dopisywanie do tej tabeli zwербowanych w trakcie wyprawy nowych członków. Zmodyfikować dane relacji *Kocury* w taki sposób aby, wykorzystując dane z relacji *Ekspedycja*, przedstawiciele za trudy wyprawy zostali wynagrodzeni zwiększeniem przydziału dodatkowego myszy o 10 oraz aby jednocześnie do relacji *Kocury* zostali dopisani nowi członkowie.

```
SQL> MERGE INTO Kocury K1
  2  USING (SELECT ksywa, imie, sex FROM Ekspedycja) K2
  3  ON (K1.pseudo=K2.ksywa)
  4  WHEN MATCHED THEN
  5      UPDATE SET myszy_extra=NVL(myszy_extra,0)+10
  6  WHEN NOT MATCHED THEN
  7      INSERT (K1.pseudo, K1.imie, K1.plec, K1.w_stadku_od,
  8              K1.szef)
  9      VALUES (K2.ksywa, K2.imie, K2.sex, SYSDATE, 'TYGRYS');
```

Scalono 3 wiersze(-y).

```
SQL> ROLLBACK;
```

Wycofywanie zostało zakończone.

```
SQL>
```

W powyższym kodzie po klauzuli **MERGE** wymieniono nazwę relacji modyfikowanej. Po klauzurze **USING** określono źródło danych w postaci wyniku podzapytania. Klauzula **ON** definiuje warunek operacji **MERGE** sprawdzany w klauzurze **WHEN**. Jeśli warunek jest spełniony (kot z relacji *Ekspedycja* jest przedstawicielem stada), wykonywana jest operacja po klauzurze **MATCHED**, jeśli nie (kot z relacji *Ekspedycja* jest kotem zwербowanym), wykonywana jest operacja po **NOT MATCHED**. Od wersji Oracle 10i po klauzurze **UPDATE** polecenia **MERGE** można dodatkowo specyfikować polecenie **DELETE** usuwające wiersze z relacji docelowej spełniające zadany w ramach **DELETE** warunek. Przykładowo dodany element

```
DELETE WHERE (myszy_extra>10)
```

usunie dodatkowo z relacji *Kocury* przedstawicieli stada o zmodyfikowanym dodatkowym przydziale myszy większym od 10.

2.5. Perspektywy (widoki)

Perspektywy są kolejnym obiektem bazy danych tworzonym i usuwanym w ramach składowej DDL języka SQL.

Perspektywa (widok) - wybór danych i wyrażeń zbudowanych na podstawie danych pobranych z jednej lub kilku relacji i z innych perspektyw

Perspektywy są widoczne przez użytkownika jako tabele a w bazie danych pamiętane są w postaci ich definicji (nie dotyczy to tzw. perspektyw zmaterializowanych). Każde odwołanie się do perspektywy powoduje jej budowę.

Perspektywy definiowane są z następujących powodów:

- aby ograniczyć dostęp do wszystkich danych relacji,
- aby ułatwić użytkownikowi pobieranie rezultatów skomplikowanych zapytań za pomocą prostych zapytań opartych na perspektywach,
- aby zwolnić użytkownika od wnikania w struktury danych,
- aby udostępnić informację widzianą w różny sposób przez różnych użytkowników.

Wybór i definicja perspektyw jest elementem fizycznego projektowania bazy danych.

Perspektywy można podzielić się na **proste** i **złożone**. Perspektywy proste to takie, w których polecenie SELECT posiada następujące cechy:

- nie zawierają złączeń,
- nie zawierają porządkowania, kwalifikatora DISTINCT, klauzuli GROUP BY, funkcji grupowych lub analitycznych,
- nie zawierają podzapytań skorelowanych lub podzapytań w klauzurze SELECT,
- nie zawierają klauzul CONNECT BY i START WITH.

Wszystkie pozostałe perspektywy nazywane są złożonymi. Istotą perspektywy prostej jest to, że można poprzez nią (w przeciwieństwie do perspektywy złożonej) wykonywać operacje DML na relacji, na bazie której perspektywa jest zbudowana. Istnieją jednak wyjątki od zakazu modyfikacji poprzez perspektywy złożone zawierające złączenie, stąd pojawiło się pojęcie **perspektywy modyfikowalnej** (różnej od perspektywy złożonej) czyli takiej, poprzez którą można wykonywać operacje DML na relacjach ją budujących. Perspektywami modyfikowalnymi nazywa się więc perspektywy, które:

- nie zawierają złączeń, za wyjątkiem takich złączeń, dla których relacje łączone zachowują klucz (*key-preserved tables*) w perspektywie. Jeśli perspektywa zawiera złączenie, dla którego relacje łączone zachowują klucz to operacja DML na perspektywie musi dotyczyć tylko takiej jednej relacji. Dodatkowo:
 - dla wykonania operacji INSERT perspektywa musi wybierać atrybuty klucza głównego i wszystkie atrybuty obowiązkowe relacji zachowującej klucz,
 - dla wykonania operacji UPDATE wszystkie modyfikowane atrybuty muszą pochodzić z relacji zachowującej klucz,
 - dla wykonania operacji DELETE operacja złączenia może dotyczyć tylko jednej relacji zachowującej klucz.
- nie zawierają porządkowania, kwalifikatora DISTINCT, klauzuli GROUP BY, funkcji grupowych lub analitycznych,
- nie zawierają podzapytań skorelowanych i podzapytań w klauzurze SELECT
- nie zawierają klauzul CONNECT BY i START WITH,
- nie zawierają wyrażeń lub pseudokolumn (polecenia INSERT i UPDATE na perspektywie nie mogą ich dotyczyć),

Relacja zachowuje klucz (*key-preserved table*) w perspektywie, jeśli każdy atrybut unikalny relacji jest też unikalny w ramach perspektywy.

Podstawowa składnia polecenia definiującego perspektywę jest następująca:

```
CREATE VIEW Nazwa_perspektywy  
                [( { atrybut_perspektywy [, ...] } )]  
AS polecenie_SELECT  
[WITH CHECK OPTION [CONSTRAINT nazwa_ograniczenia]] |  
[WITH READ ONLY]
```

gdzie atrybuty perspektywy są nowymi nazwami wyrażeń (atrybutów) wyspecyfikowanych w klauzurze SELECT polecenia SELECT (lista ta nie musi wystąpić jeśli wszystkie wyrażenia są atrybutami relacji/perspektywy; wtedy perspektywa będzie posiadać atrybuty o nazwach takich samych jak odpowiednie atrybuty relacji/perspektywy, z której pobierane są dane), opcjonalna klauzula WITH CHECK OPTION (dotycząca tylko perspektyw prostych) dopuszcza aktualizację poprzez perspektywę pod warunkiem, że zmienione lub nowe rekordy nadal będą się pojawiały w widoku (będą spełniały warunek po klauzurze WHERE ciała polecenia SELECT). Po klauzurze WITH CHECK OPTION można dodatkowo umieścić klauzulę CONSTRAINT nazwa_ograniczenia umożliwiającą nadanie nazwy tworzonemu ograniczeniu. Opcjonalna klauzula WITH READ ONLY (dotycząca tylko perspektyw prostych) uniemożliwia aktualizowania poprzez widok.

Zad. Zdefiniować perspektywę prostą udostępniającą część danych (pseudonim, data wstąpienia do stada, przydział myszy) kotów należących do bandy nr 3.

```
SQL> CREATE VIEW Banda3
  2  AS
  3  SELECT pseudo,w_stadku_od,przydział_myszy
  4  FROM Kocury
  5  WHERE nr_bandy=3;
```

Perspektywa została utworzona.

SQL>

Zdefiniowana perspektywa jest perspektywą prostą więc możliwa jest poprzez nią modyfikacja danych w relacji Kocury.

```
SQL> UPDATE Banda3
  2  SET przydział_myszy=55
  3  WHERE pseudo='ZERO';
```

1 wiersz został zmodyfikowany.

SQL>

To, że dane rzeczywiście zostały zmodyfikowane pokazuje poniższe zapytanie.

```
SQL> SELECT pseudo,przydział_myszy
  2  FROM Kocury
  3  WHERE pseudo='ZERO';
```

PSEUDO	PRZYDZIAŁ_MYSZY
ZERO	55

```
SQL> ROLLBACK;
```

Wycofywanie zostało zakończone.

SQL>

Polecenie ROLLBACK jest elementem składowej DCL języka SQL i służy do wycofania realizowanej transakcji (w powyższym przypadku transakcja składała się z jednego polecenia UPDATE).

Zad. Zdefiniować perspektywę złożoną udostępniającą nazwy band oraz minimalny, maksymalny i średni przydział myszy w każdej bandzie.

```
SQL> CREATE VIEW Myszki_w_bandach (nazwa,minim,maksim,srednio)
  2 AS
  3 SELECT nazwa,MIN(przydzial_myszy),MAX(przydzial_myszy),
  4         AVG(przydzial_myszy)
  5 FROM Kocury,Bandy
  6 WHERE Kocury.nr_bandy=Bandy.nr_bandy
  7 GROUP BY nazwa;
```

Perspektywa została utworzona.

```
SQL> SELECT * FROM Myszki_w_bandach;
```

NAZWA	MINIM	MAKSIM	SREDNIO
SZEFOSTWO	22	103	50
BIALI LOWCY	20	75	49,75
CZARNI RYCERZE	24	72	56,8
LACIACI MYSLIWI	40	65	49,4

```
SQL>
```

Perspektywy proste z klauzulą WITH CHECK OPTION mogą być narzędziem do wprowadzania dodatkowych ograniczeń na dane.

Zad. Zdefiniować perspektywę udostępniającą część danych kotów z bandy nr 4 umożliwiającą operacje DML na perspektywie tylko dla tej bandy.

```
SQL> CREATE VIEW Banda4
  2 AS
  3 SELECT pseudo,imie,funkcja,przydzial_myszy,nr_bandy
  4 FROM Kocury WHERE nr_bandy=4
  5 WITH CHECK OPTION;
```

Perspektywa została utworzona.

```
SQL>
```

Przykładowa niemożliwa operacja:

```
SQL> INSERT INTO Banda4 VALUES ('LALA', 'KOBZA', 'KOT', 30, 3);  
INSERT INTO Banda4 VALUES ('LALA', 'KOBZA', 'KOT', 30, 3)
```

*

BŁĄD w linii 1:

ORA-01402: naruszenie klauzuli WHERE dla perspektywy z WITH
CHECK OPTION

SQL>

Zad. Po długim zastanowieniu Tygrys doszedł do wniosku, że sytuacja w stadzie jest równie doskonała jak i on jest doskonały. Nakazał więc nadwornemu informatykowi zdefiniowanie perspektywy „pilnującej” aby nie można było naruszyć istniejącego status quo. Perspektywa miała więc dbać o to aby:

- nie mogła powstać żadna nowa banda,
- żaden kot spoza elity szefów nie uzurpował sobie prawa do bycia szefem,
- koty mogły pełnić tylko funkcje już istniejące,
- przydział myszy był przynajmniej równy kociemu minimum socjalnemu (6 myszy) oraz żeby nie mógł przekroczyć przydziału myszy Tygrysa.

Zdefiniować perspektywę spełniającą te wymagania.

```
SQL> CREATE OR REPLACE VIEW Kocie_status_quo  
2 AS  
3 SELECT pseudo, imie, szef, funkcja, przydział_myszy, nr_bandy  
4 FROM Kocury  
5 WHERE (nr_bandy IN (SELECT nr_bandy FROM Kocury)  
6          OR nr_bandy=5 OR nr_bandy IS NULL)  
7        AND (szef IN (SELECT szef FROM Kocury)  
8          OR szef IS NULL)  
9        AND (funkcja IN (SELECT funkcja FROM Kocury)  
10         OR funkcja='HONOROWA' OR funkcja IS NULL)  
11        AND (przydział_myszy BETWEEN 6  
12              AND (SELECT przydział_myszy  
13                   FROM Kocury  
14                   WHERE pseudo='TYGRYS'))  
15         OR przydział_myszy IS NULL)  
16        AND pseudo<>'TYGRYS'  
17 WITH CHECK OPTION CONSTRAINT nic_wiecej;
```

Perspektywa została utworzona.

SQL>

Jak już wcześniej wspomniano, operacje DML *via* perspektywa są dozwolone tylko dla perspektyw prostych i modyfikowalnych. W pełni poprzez te perspektywy można wykonywać jedynie operację DELETE. Operacja UPDATE wymaga już spełnienia dodatkowego warunku: atrybut perspektywy nie może być zdefiniowany za pomocą wyrażenia. Operacja INSERT jest jeszcze bardziej wymagająca: oprócz spełnienia wszystkich wcześniejszych warunków wymagane jest aby atrybuty obowiązkowe były wybierane przez perspektywę.

Perspektywa usuwana jest poleceniem DROP VIEW o następującej składni:

DROP VIEW Nazwa_perspektywy

2.6. Słownik bazy danych Oracle

Większość baz danych posiada tzw. metabazę czyli bazę danych, która zawiera dane o bazie. Metabaza nazywana jest też słownikiem bazy danych.

Słownik bazy danych jest to zestaw relacji i perspektyw systemowych w których znajdują się informacje o bazie danych. Informacje te (tylko perspektywy systemowe) dostępne są tylko do czytania.

W słowniku można znaleźć między innymi:

- nazwy użytkowników,
- prawa i uprawnienia przyznane użytkownikom,
- nazwy i opisy obiektów bazy danych (relacje, perspektywy, indeksy, synonimy, sekwencje itp.),
- więzy związane z relacjami,
- informacje statystyczne o tym, kto i w jaki sposób pracował na obiektach bazy danych.

Nazwy perspektyw słownika odpowiadają zawartej w nich treści.
Podzielone są one na trzy grupy wg przedrostków USER, ALL, DBA

USER_xxxx - informacje o obiektach będących własnością
użytkownika (np. USER_OBJECTS)

ALL_xxxx - informacje o wszystkich obiektach, do których
użytkownik posiada dostęp (np. ALL_VIEWS)

DBA_xxxx - informacje o wszystkich obiektach (tylko dla
administratora bazy danych)

Istnieją także perspektywy bez specjalnego przedrostka np.

DICTIONARY - lista wszystkich relacji, perspektyw i synonimów
rozszerzona o komentarze

DICT_COLUMNS - lista atrybutów należących do dostępnych dla
użytkownika obiektów

Najważniejsze (najczęściej używane) perspektywy słownikowe:

Nazwa	Synonim	Opis
DICTIONARY	DICT	Wszystkie obiekty bazy
USER_OBJECTS	OBJ	Obiekty użytkownika
USER_CATALOG	CAT	Relacje, perspektywy, synonimy i sekwencje dostępne dla użytkownika
USER_TABLES	TABS	Opisy relacji użytkownika
USER_TAB_COLUMNS	COLS	Atrybuty z relacji i perspektyw użytkownika

USER_COL_COMMENTS		Komentarze do atrybutów relacji i perspektyw użytkownika
USER_TAB_COMMENTS		Komentarze do relacji i perspektyw użytkownika
USER_SEQUENCES	SEQ	Opisy sekwencji użytkownika
USER_SYNONYM	SYN	Opisy synonimów użytkownika
USER_VIEWS		Definicje perspektyw użytkownika
USER_INDEXES	IND	Opisy indeksów użytkownika
ALL_OBJECTS		Obiekty dostępne dla użytkownika
ALL_TAB_COLUMNS		Atrybuty wszystkich relacji i perspektyw dostępnych dla użytkownika

2.7. SQL do generowania SQL

Bazując na informacji ze słownika bazy danych można, stosując polecenia SQL, wygenerować inne polecenia SQL. SQL z SQL buduje się aby:

- uniknąć powtarzającego się kodowania,
- uzyskać pomocniczą informację ze słownika bazy danych,
- odtworzyć relacje, perspektywy, indeksy.

Zad. Wykorzystując informacje zawarte w słowniku bazy danych stworzyć skrypt generujący, w ramach środowiska SQL*Plus, schematy wszystkich relacji należących do aktualnego użytkownika.

```
SELECT 'DESC '||object_name
FROM User_objects
WHERE object_type='TABLE';
```

```
'DESC' || OBJECT_NAME
```

```
-----
DESC BANDY
DESC FUNKCJE
DESC KOCURY
DESC WROGOWIE
DESC WROGOWIE_KOCUROW
```

} Jeśli udałoby się ten fragment skierować do pliku, uzyskaliby się skrypt generujący opisy relacji

```
SET HEADING OFF ----- Odwołanie wyświetlania nagłówków
```

```
SET PAGESIZE 0 ----- Odwołanie łamania stron
```

```
SET FEEDBACK OFF ----- Odwołanie wyświetlania zwrotnych
                          informacji diagnostycznych
```

```
SET TERMOUT OFF ----- Odwołanie wyświetlania na ekran
```

```
SPOOL C:\co_mam ----- Przekierowanie wyświetlanych
                          informacji do pliku
```

```
SELECT 'DESC '||table_name
FROM User_tables;
```

```
SPOOL OFF
```

```
SET HEADING ON
```

```
SET PAGESIZE 24
```

```
SET FEEDBACK ON
```

```
SET TERMOUT ON
```

```
START C:\co_mam.lst
```

2.8. Indeksy

Indeksy nie należą do standardu ANSI SQL choć są implementowane przez większość SZBD, także przez Oracle. Ich tworzenie i usuwanie jest kolejnym elementem składowej DDL języka SQL.

Indeks – struktura danych służąca do:

- przyspieszania wybierania krotek na podstawie wyspecyfikowanych atrybutów relacji,
- wymuszania unikatowych wartości atrybutu.

Podstawowy typ indeksów są zwykle tworzone w ramach SZBD za pomocą tzw. zrównoważonych B-drzew (indeksy B*-tree). Gwarantuje to, w przybliżeniu, te same czasy dostępu do wszystkich krotek relacji, niezależnie od ich umiejscowienia w relacji. W niektórych systemach, także w Oracle, dla więzów spójności UNIQUE oraz PRIMARY KEY indeksy tworzone są niejawnie (automatycznie). Do jawnego tworzenia indeksów służy polecenie CREATE INDEX o następującej składni:

```
CREATE [UNIQUE] INDEX Nazwa_indeksu  
ON Nazwa_relacji({atrybut [DESC | ASC] [, ...]})
```

gdzie wystąpienie UNIQUE powoduje wymuszenie unikalności dla zbioru wartości atrybutów wymienionych po nazwie relacji a DESC określa malejący kierunek budowy kolumny indeksu (domyślnie kolumna indeksu budowana jest rosnąco – ASC) *via* atrybut indeksu. Na atrybut indeksu może być dodatkowo nałożona funkcja. Indeks taki nazywany jest indeksem funkcyjnym.

Można wyróżnić dwa rodzaje indeksów:

- indeks prosty: oparty na jednym atrybucie,
- indeks złożony: oparty na więcej niż jednym atrybucie.

Rozróżnienie to może mieć konsekwencje w algorytmach optymalizujących działanie poleceń. Przykładowo w Oracle tylko indeksy proste są scalane tzn. wykorzystywane razem.

O użyciu lub nie indeksu może decydować SZBD na podstawie analizy wykonanej przez narzędzie optymalizujące działanie zapytania (jeśli system w taki optymalizator jest wyposażony - Oracle takie posiada).

Aby indeks mógł zostać użyty, muszą być spełnione następujące warunki:

- indeksowany atrybut musi wystąpić w klauzurze WHERE,
- atrybut w klauzurze WHERE nie może być argumentem funkcji lub częścią wyrażenia.

Wybór i definicja indeksów jest elementem projektowania fizycznego bazy danych. Przy ich wyborze należy się kierować następującymi zasadami:

- założenie indeksu jest korzystne dla relacji z większą liczbą krotek (w Oracle ponad 200). Przy małych relacjach czas odczytu indeksu może być większy od zysku czasu wykonania polecenia,
- wskazana jest indeksacja ze względu na atrybuty, których wartości raczej się nie powtarzają,
- wskazana jest indeksacja ze względu na atrybuty często używane w klauzurze WHERE (także w warunkach łączących),
- jeśli dwa lub więcej atrybutów występuje często razem w klauzurze WHERE, należy tworzyć indeksy złożone,
- należy unikać więcej niż trzech indeksów dla jednej relacji (przeciążenie operacji DML). Reguła ta nie dotyczy przypadku, gdy najczęściej używanym poleceniem jest SELECT,
- przy wsadowej modyfikacji relacji wskazane jest czasowe usunięcie indeksów nałożonych na tą relację ponieważ każde polecenie wsadowe powoduje niezależnie odświeżenie indeksu co zajmuje czas. Po modyfikacji wsadowej należy usunąć indeksy przywrócić.

Indeksy usuwane są poleceniem DROP INDEX o następującej składni:

DROP INDEX Nazwa_indeksu

2.9. Sekwencje

Kolejnym przydatnym obiektem bazy danych tworzonym, modyfikowanym i usuwanym za pomocą poleceń składowej DDL dialektu Oracle SQL'a są sekwencje

Sekwencja to obiekt bazy danych służący do generowania unikalnych wartości, zwykle na potrzeby kluczy głównych.

Do tworzenia, modyfikowania i usuwania sekwencji służą odpowiednio polecenia DDL: CREATE, ALTER i DROP o składni:

```
CREATE SEQUENCE Nazwa_sekwencji  
[START WITH wartość_początkowa]  
[INCREMENT BY krok]  
[MAXVALUE wartość_maksymalna | NOMAXVALUE]  
[MINVALUE wartość_minimalna | NOMINVALUE]  
[CYCLE | NOCYCLE];
```

```
ALTER SEQUENCE Nazwa_sekwencji  
[INCREMENT BY krok]  
[MAXVALUE wartość_maksymalna | NOMAXVALUE]  
[MINVALUE wartość_minimalna | NOMINVALUE]  
[CYCLE | NOCYCLE];
```

```
DROP SEQUENCE Nazwa_sekwencji;
```

gdzie **START WITH** określa pierwszą liczbę jaka będzie zwracana przez sekwencję, **INCREMENT BY** określa, o ile mają być zwiększane kolejne numery (domyślnie 1), **MAXVALUE** i **MINVALUE** określają górną i dolną granicę sekwencji (domyślne wartości odpowiednio **NOMAXVALUE** i **NOMINVALUE**), **CYCLE** określa, czy sekwencja powinna być tworzona cyklicznie (domyślnie **NOCYCLE**) od wartości **MINVALUE** do **MAXVALUE** (odwrotnie dla sekwencji malejącej)

Do odczytu aktualnej wartości sekwencji wykorzystywany jest pseudoatrybut CURRVAL.

```
SELECT nazwa_sekwencji.CURRVAL FROM Dual;
```

Zwiększenie wartości sekwencji uzyskuje się poprzez wykorzystanie pseudoatrybutu NEXTVAL.

```
SELECT nazwa_sekwencji.NEXTVAL FROM Dual;
```

Uwaga:

Wielokrotne odwołanie się do NEXTVAL w tym samym poleceniu powoduje zwrócenie tego samego numeru.

Zad. Stworzyć sekwencję, która zapewni wpisywanie do relacji Bandy nowych band o kolejnych numerach od 6 począwszy.

```
SQL> CREATE SEQUENCE Numery_band  
2 START WITH 6;
```

Sekwencja została utworzona.

```
SQL> INSERT INTO Bandy  
2 VALUES (Numery_band.NEXTVAL, 'NOWORYSZE', 'LASEK');
```

1 wiersz został utworzony.

```
SQL> ROLLBACK;
```

Wycofywanie zostało zakończone.

```
SQL> DROP SEQUENCE Numery_band;
```

Sekwencja została usunięta.

```
SQL>
```


Pseudoatrybuty CURRVAL i NEXTVAL mogą być stosowane:

- w klauzuli SELECT polecenia SELECT,
- w liście wartości rozkazu INSERT,
- w klauzuli SET polecenia UPDATE
- tylko w zapytaniu głównym (najbardziej zewnętrznym).

Pseudoatrybuty CURRVAL i NEXTVAL nie mogą być stosowane:

- w klauzuli SELECT definiującej perspektywę,
- z kwalifikatorem DISTINCT,
- jeśli występują klauzule ORDER BY, GROUP BY, CONNECT BY, HAVING,
- z operatorami UNION, INTERSECT, MINUS
- w podzapytaniach.

2.10. Synonimy

Synonimy to kolejne obiekty bazy danych tworzone i usuwane za pośrednictwem składowej DDL dialektu Oracle SQL'a.

Synonimy są to alternatywne nazwy dla niektórych obiektów bazy danych (np. tabel, sekwencji, procedur czy pakietów)

Do tworzenia i usuwania synonimów służą odpowiednio polecenia DDL: CREATE i DROP o składni:

CREATE [**PUBLIC**] **SYNONYM** Nazwa_synonimu **FOR** obiekt;

DROP [**PUBLIC**] **SYNONYM** Nazwa_synonimu;

Zad. *Stworzyć dla użytkownika Chytry synonim do perspektywy o nazwie Kocie_status_quo.*

```
SQL> CREATE SYNONYM Ksq FOR Chytry.Kocie_status_quo;
```

Synonim został utworzony.

```
SQL> DROP SYNONYM Ksq;
```

Synonim został usunięty.

```
SQL>
```

2.11. Klastry

Klastry są alternatywą wobec relacji formą składowania danych w bazie danych. Wykorzystywane są one dla tabel, które są bardzo często łączone operacją równozłączenia. Dzięki zapisowi tabel w klastrze łączone tabele pamiętane są strukturze klastra w tym samym obszarze dysku co pozwala uniknąć kosztownych złączeń a więc i zdecydowanie skrócić czas dostępu do danych z tabel. Klaster posiada swój klucz, którym zazwyczaj jest atrybut definiujące złączenie. Klaster w swej podstawowej wersji jest tworzony zgodnie ze składnią:

```
CREATE CLUSTER Nazwa_klastra  
({nazwa_atrybutu_klucza typ_atrybutu_klucza [, ...]});
```

gdzie nazwa atrybutu klucza jest dowolną nazwą a typ atrybutu klucza jest zgodny z typem atrybutu definiującego złączenie dla tabel, które zostaną umieszczone w klastrze. Po stworzeniu klastra należy umieścić tabele w klastrze. Odbywa się to zgodnie ze składnią:

```
Polecenie_CREATE_TABLE  
CLUSTER Nazwa_klastra ({atrybut_złączenia [, ..]});
```

Po dodaniu do klastra tabel a przed wprowadzeniem do niego danych należy utworzyć indeks klastra zgodnie ze składnią:

```
CREATE INDEX Nazwa_indeksu  
ON CLUSTER Nazwa_klastra;
```

Do tabel tak utworzonego klastra można za pomocą polecenia INSERT wprowadzić dane oraz następnie definiować dla nich zapytania SELECT.

Zad. Zakładając, że często łączonymi relacjami są uproszczona do atrybutów imię i pseudo relacja Kocury oraz relacja Bandy stworzyć klaster zawierający te relacje.

```
SQL> CREATE CLUSTER Koty_bandy  
2  (nr_bandy NUMBER(2));
```

Klaster został utworzony.

```
SQL> CREATE TABLE Bandy1  
2  (nr_bandy NUMBER(2) CONSTRAINT nr_bandy1_pk PRIMARY KEY,  
3   nazwa VARCHAR2(20) NOT NULL,  
4   teren VARCHAR2(15) UNIQUE NOT NULL)  
5  CLUSTER Koty_bandy (nr_bandy);
```

Tabela została utworzona.

```
SQL> CREATE INDEX Koty_bandy_ind  
2  ON CLUSTER Koty_bandy;
```

Indeks został utworzony.

```
SQL> CREATE TABLE Kocury1  
2  (imie VARCHAR2(15) NOT NULL,  
3   pseudo VARCHAR2(15) CONSTRAINT pseudo_pk1 PRIMARY KEY,  
4   nr_bandy NUMBER(2) CONSTRAINT nr_bandy_fk1  
5   REFERENCES Bandy1(nr_bandy))  
6  CLUSTER Koty_bandy (nr_bandy);
```

Tabela została utworzona.

SQL>

Innym typem klastra jest tzw. klaster haszowy. Informacje o nim zostaną przedstawione w ramach tematu dotyczącego optymalizatorów zapytań.

2.12. Przetwarzanie transakcji i transakcje współbieżne

Polecenia służące do zarządzania transakcjami należą do składowej DCL języka SQL.

Transakcja to kończąca się powodzeniem lub niepowodzeniem, złożona z szeregu zmian w jednej lub więcej relacji operacja na bazie danych

Dwa podstawowe polecenia składowej DCL języka SQL to polecenie COMMIT zatwierdzające jawnie transakcję i polecenie ROLLBACK wycofujące jawnie transakcję.

Można wyróżnić dwa typy transakcji:

- transakcje DDL - równoważne pojedynczym rozkazom DDL,
- transakcje DML - złożone z dowolnej liczby operacji DML

Każda transakcja ma swój początek i swój koniec.

Początek transakcji - pierwszy wykonywalny rozkaz DML lub DDL

Koniec transakcji - wystąpienie jednego z poniższych przypadków

- ◆ polecenie COMMIT (jawne zatwierdzenie transakcji) lub ROLLBACK (jawne wycofanie transakcji),
- ◆ rozkaz DDL
- ◆ pewien typ błędu (np. zakleszczenie - "dead-lock"),
- ◆ zakończenie sesji programu,
- ◆ awaria komputera

Oprócz jawnego zatwierdzenia lub wycofania transakcji przez użytkownika system może zatwierdzić lub wycofać transakcję niejawnie.

Transakcja jest zatwierdzana niejawnie:

- ◆ przed rozkazem DDL,
- ◆ po rozkazie DDL,
- ◆ po normalnym odłączeniu od bazy.

Transakcja jest wycofana niejawnie:

- ◆ po wystąpieniu błędu systemu.

Rozkaz z błędem w transakcji DML usuwany jest automatycznie bez straty wcześniejszych zmian w transakcji (mechanizm "STATEMENT LEVEL ROLLBACK" oparty na tworzeniu przed każdym rozkazem DML systemowych punktów zachowania). Punkty zachowania umożliwiają wydzielenie części transakcji z możliwością wycofania tylko tej części. Punkt zachowania jawnie tworzony jest zgodnie ze składnią:

SAVEPOINT nazwa_punktu;

Wycofanie części transakcji do punktu zachowanie (bez zamknięcia) realizuje polecenie:

ROLLBACK TO SAVEPOINT nazwa_punktu;

Polecenia transakcji DML mogą być niejawnie zatwierdzane w trakcie transakcji. Służy do tego polecenie SET AUTOCOMMIT o składni:

SET AUTOCOMMIT {ON | OFF | liczba_poleceń}

Po wykonaniu powyższego rozkazu polecenia transakcji DML będą niejawnie zatwierdzane co określoną w powyższej składni liczbę poleceń. W przypadku wystąpienia elementu ON niejawnie zatwierdzenie nastąpi po każdym poleceniu DML (liczba_poleceń=1).

Transakcje mogą być realizowane w kilku trybach. Tryb pracy transakcji w SZBD Oracle ustalany jest poleceniem SET TRANSACTION o składni:

SET TRANSACTION

{READ {ONLY | WRITE}} |
{ISOLATION LEVEL {SERIALIZABLE |
READ COMMITED}}

gdzie ustawienie:

READ ONLY	oznacza, że jedynym możliwym rozkazem jest SELECT (obraz bazy danych z momentu rozpoczęcia transakcji).
READ WRITE	oznacza, że transakcja może czytać i zapisywać dane (przeciwieństwo READ ONLY). Jest to tryb domyślny.
ISOLATION LEVEL SERIALIZABLE	oznacza, że jest możliwa modyfikacja danych pod warunkiem, że podczas transakcji modyfikowane dane nie były zmieniane przez inne transakcje (obraz bazy danych z momentu rozpoczęcia transakcji).
ISOLATION LEVEL READ COMMITED	oznacza, że możliwa modyfikacja danych (dane zmieniane przez inne transakcje są widoczne po zatwierdzeniu tych transakcji). Jest to tryb domyślny.

2.12.1. Transakcje współbieżne

Jednym z głównych zadań systemu zarządzania danymi jest zorganizowanie współbieżnej pracy wielu użytkowników, tak aby zachować spójność danych. Oracle rozwiązuje ten problem stosując system blokowania dostępu do danych.

Można wyróżnić dwa rodzaje blokowania:

- blokowanie DDL (blokowanie słownika systemowego) - realizowane automatycznie (niejawnie) przez system zarządzania bazą,
- blokowanie DML (blokowanie bazy użytkownika) - realizowane automatycznie (niejawnie) przez system zarządzania bazą. Użytkownik może jednak założyć indywidualną blokadę stosując rozkaz SQL.

Jawnie blokowanie można wykonać na poziomie wierszy i na poziomie relacji. Wiersze relacji mogą być jawnie blokowane poleceniem SELECT z klauzulą FOR UPDATE o składni:

polecenie_SELECT
FOR UPDATE [NOWAIT];

gdzie opcjonalna klauzula NOWAIT powoduje anulowanie polecenia blokady w przypadku gdy wiersze wybierane przez polecenie SELECT są zablokowane przez inną transakcję. Blokada jest zwalniana z chwilą zatwierdzenia lub wycofania transakcji. W polecenia SELECT z klauzulą FOR UPDATE nie wolno używać następujących elementów:

- DISTINCT,
- GROUP BY,
- operatorów zbiorowych,
- funkcji grupowych.

Zad. Zablokować do poprawy wszystkie krotki relacji Kocury opisujące koty płci żeńskiej.

```
SQL> SELECT *  
      2 FROM Kocury  
      3 WHERE plec='D'  
      3 FOR UPDATE NOWAIT;
```

Powyższe polecenie nie zablokuje wierszy w przypadku ich wykorzystywania przez inną transakcję.

Jawna blokada może być także wykonywana na poziomie relacji. Służy do tego polecenie LOCK TABLE o składni:

LOCK TABLE nazwa_relacji **IN** nazwa_trybu **MODE** [NOWAIT];

gdzie trybem blokady może być ROW SHARE (współdzielenie wierszy), ROW EXCLUSIVE (blokada wierszy), SHARE (współdzielenie relacji) lub EXCLUSIVE (blokada relacji).

2.12.2. Zakleszczenie (Dead-lock)

Kilka równoległych transakcji może doprowadzić do sytuacji tzw. zakleszczenia (transakcje blokują sobie wzajemnie zasoby). Oracle automatycznie wykrywa sytuację zakleszczenia i likwiduje ją poprzez przerwanie blokującego polecenia jednej z transakcji. Poniżej przedstawiono przykład zakleszczenia oraz reakcję systemu Oracle a także sposób wyjścia z zakleszczenia.

Transakcja 1

```
SQL> UPDATE Kocury SET myszy_extra=50 WHERE pseudo='TYGRYS';  
1 wiersz został zmodyfikowany.
```

Transakcja 2

```
SQL> UPDATE Kocury SET przydzial_myszy=60 WHERE pseudo='BOLEK';  
1 wiersz został zmodyfikowany.
```

Transakcja 1

```
SQL> UPDATE Kocury SET myszy_extra=70 WHERE pseudo='BOLEK';
```

System czeka na zwolnienie blokady przez Transakcję 2 (krotka BOLKA).

Transakcja 2

```
SQL> UPDATE Kocury SET przydział_myszy=120  
2 WHERE pseudo='TYGRYS';
```

System czeka na zwolnienie blokady przez Transakcję 1 (krotka TYGRYSA).

Transakcja 1

```
UPDATE Kocury  
*
```

```
BŁĄD w linii 1:  
ORA-00060: deadlock detected while waiting for resource
```

System przerywa działanie polecenia lecz blokada krotki TYGRYSA jest nadal nie usunięta.

Transakcja 2

Czeka na zwolnienie blokady przez Transakcję 1 (krotka TYGRYSA).

Transakcja 1

```
ROLLBACK;
```

Wycofanie zostało zakończone.

Wycofanie transakcji = zwolnienie blokady.

Transakcja 2

1 wiersz został zmodyfikowany.

Dzięki współpracy z systemem użytkownicy mogą wyjść z sytuacji zakleszczenia.

