Uruchomienie SQL*Plus:

sqlplus [użytkownik [/ hasło]]

Zasady wpisywania komend PL/SQL:

- komendy mogą być wieloliniowe
- nie można skracać słów kluczowych
- można oddzielać słowa spacjami lub tabulacją
- słowa muszą zawierać się całkowicie w jednej linii (nie można dzielić słów)
- w słowach kluczowych duże i małe litery nie są rozróżniane

- ostatnio wprowadzona komenda jest pamiętana w buforze SQLa, można ją krotnie wykonywać, edytować, etc.
  Koniec wpisywania do bufora - po wprowadzeniu pustej linii.
- komenda jest uruchamiana jeżeli została zakończona znakiem '**;**' lub '/'
- komendę z bufora SQL można uruchomić komendą '**run**' lub wpisując '/' jako pierwszy znak w linii (a potem - Enter)

- łańcuchy znaków ujmowane w apostrofy, dwa apostrofy oznaczają apostrof w łańcuchu znaków
- stałe tekstowe i stałe typu *data* są ujmowane w apostrofy lub cudzysłowy - jeżeli zaczynamy od apostrofu musimy nim zakończyć stałą

- istnieje wyróżniona wartość **NULL** używana do wpisywania 'wartości niezdefiniowanej' - jest wyświetlana jako puste miejsce. To nie jest zero - wyrażenia w których występuje obiekt typu **null** również dają wynik typu **NULL**. Istnieje funkcja - **NVL** - którą można skonwertować wartości **NULL** do dowolnych wybranych wartości

- w skryptach SQL zakończenie linii myślnikiem powoduje, że następna linia jest traktowana jako linia kontynuacji. Wpisanie / w ostatniej linii skryptu spowoduje - po wczytaniu - jego wykonanie.

Edycja bufora SQL*Plus:

Wszystkie komendy działają standardowo tylko w bieżąco wybranej linii z bufora SQL*Plusa.

numer	linia o podanym numerze staje się linią bieżącą
a tekst	dopisz tekst na końcu bieżącej linii
c/stary/nowy/	zamień teksty
cl buff	wyczyść bufor
del	skreśl bieżącą linię
i tekst ...	wstaw w bieżącym miejscu dowolną ilość znaków (linii tekstu)
list	listuj wszystkie linie w buforze
list od,do	listuj tylko linie o podanych numerach
r	uruchom komendę z bufora

Polecenia 'pomocnicze':

save fname	zapisz zawartość bufora we wskazanym pliku
get fname	wczytaj komendy ze wskazanego pliku
start fname	wykonaj komendy SQLa zapisane na wskazanym pliku
ed fname	edytuj wskazany plik
spool fname	pamiętaj 'historię' (wpisane komendy i ich wyniki) we wskazanym pliku
desc table_name	wyświetl opis pól wskazanej tabeli
help	wiadomo
conn użytkownik/hasło	zmień (dołącz się) bieżącego użytkownika
exit	zgodnie z oczekiwaniami
prompt tekst	wyświetl *tekst* na ekranie. Używane w skryptach do wypisywania komunikatów

Typy danych:

- **CHAR(w)**

  łańcuchy znaków, do 255 elementów, stałej długości, uzupełniany spacjami przy porównaniach

- **VARCHAR(w)**

  łańcuchy znaków, do 255 elementów, zmiennej długości

- **VARCHAR2(w)**

  zmienna znakowa, do 2 000 znaków, w *Oracle 7* zmienne **VARCHAR** i **VARCHAR2** są traktowane jako zmienne tego samego typu

- **NUMBER**

  liczby zmiennoprzecinkowe, maks. 38 cyfr znaczących, maks. $9.99 * 10^{124}$

- **NUMBER(w)**

  jak **NUMBER**, ale ograniczone do *w* cyfr znaczących (*w* =< 38)

- **NUMBER(w,d)**

  jak **NUMBER(w)** ale dodatkowo określamy ilość cyfr po kropce dziesiętnej

- **DATE**

  data i czas, od 01/01/1472 pne do 31/12/4712 ne

- **LONG**

  tylko jedno pole tego typu może występować w rekordzie, do 2 GB.
  Ograniczone użycie - nie można używać takiego pola jako indeksu, nie można według niego sortować, nie działają na nim funkcje znakowe. Używane do przechowywania grafiki i dźwięku (multimedia).

Wybór danych:

- **SELECT**

SELECT [DISTINCT | ALL]

* | column | expr [[AS] c_alias] ["column header"], |
table.* | table.column [[AS] c_alias] ["column header"] , ...

FROM table [t_alias] ...

[WHERE condition]

[ORDER BY {expr | c_alias | position} [ASC | DESC]

[, {expr | c_alias | position}

[ASC | DESC]] ...]

DISTINCT	tylko unikalne rekordy
ALL	wszystkie rekordy
	DISTINCT może wystąpić tylko raz, bezpośrednio po słowie **select**
*	wszystkie kolumny z tablicy
column	tylko nazwane kolumny
expr [[AS] c_alias]	wartości wyrażeń, aliasów można potem użyć tylko w nagłówkach kolumn i raportów (*column, ttitle, btitle*)
table.*	wszystkie kolumny wskazanej tablicy

Po nazwie kolumny a jeszcze przed przecinkiem definiującym następną kolumnę może wystąpić stała tekstowa - stała ta definiuje wtedy nagłówek dla tej kolumny. Stałe znakowe, które mogą również wystąpić w tej części **select** są ujmowane w apostrofy o ile zwierają spacje - jeżeli są to pojedyncze słowa - mogą wystąpić bez apostrofów.

Specjalne operatory w wyrażeniach używanych do wyświetlania kolumn:

||	łączy dwie sąsiadujące kolumny, na wyjściu pokażą się jako jedna, połączona wartość

FROM table [t_alias] nazwy tablic z których wybierana będzie informacja. Nazwa tablicy może być postaci:

table
user.table
link.table

Defaultowo - bieżący użytkownik. Podanie nazwy użytkownika może przyspieszyć działania *select.*
Alias zdefiniowany w tym miejscu może być użyty do rozróżniania kolumn z różnych tablic w tym selekcie

link - dostęp do tablic w sieci: ***user.table@net_address***

[WHERE condition] do selekcji rekordów. Tylko rekordy dla których ten warunek będzie *TRUE* będą selekcjonowane. Można tu użyć normalnych, pełnych wyrażeń logicznych i relacyjnych ('*C like*')
Można normalnie użyć **not** by zanegować warunek, **and, or** i **nawiasy** również działają normalnie. Jeżeli używamy **where** musi ona wystąpić bezpośrednio po **from**. Jeżeli w wyrażeniach użytych w **where** używamy danych znakowych lub dat trzeba je ująć w pojedyncze apostrofy.

Można porównywać różne kolumny tego samego wiersza w wyrażeniu użytym w **where**. Dodatkowo można użyć operatorów SQL:

between *expr*$_1$ and *expr*$_2$	wartość należy do obustronnie domkniętego przedziału [*expr*$_1$,*expr*$_2$]
in (lista)	wartość jest jednym z elementów listy
like *wzornik*	wartość jest zgodna z podanym wzorcem (≈ wyrażeniem regularnym). Znaki specjalne: **%** dowolna ilość dowolnych znaków _ dokładnie jeden dowolny znak **%** może wystąpić zarówno na początku jak i na końcu wzornika (**%ala%**)
is null	wartość jest wartością **NULL.** Próba porównania *var != NULL* nie działa - zawsze dostaniemy wartość ***FALSE*** - do porównywania z wartością **NULL** można używać tylko 'is null' lub 'is not null'

Pozwala to na konstruowanie warunków w rodzaju:

ename like 'S%' nazwisko zaczyna się na S
ename like '____' nazwisko jest czteroliterowe

Zwykły priorytet operatorów:

max == != < > <= >= between ... and ... in like is null
not
and
min or

[ORDER BY {expr | c_alias | position} [ASC | DESC]
[, {expr | c_alias | position} [ASC | DESC]] ...]

sortuje wynik komendy **select**, musi być ostatnią w komendzie **select.** Można podać wiele nazw kolumn lub wiele wyrażeń według których odbywać się będzie sortowanie. **NULL** jest 'większe od wszystkiego'. Bez użycia '**order by**' dostajemy rekordy w przypadkowej kolejności - dwa kolejne zapytania mogą zwrócić rekordy w innej kolejności. Sortować można również według kolumn które nie są wyświetlane **select**em.

Uwagi do select'u:

- wartości **NULL** są wyświetlane jako puste miejsca. Trzeba użyć *nvl(zmienna,wartość)* aby spowodować wyświetlenie 'wartości'.

Ćwiczenia:

- wyświetl wszystkie dane z tablicy EMP
- wybierz dane o pracownikach których zarobki mieszczą się w przedziale [1000,2000]. Wyświetl tylko nazwisko, numer działu i zarobki.
- wyświetl nazwy unikalnych stanowisk pracy, uporządkowane alfabetycznie (EMP)
- wyświetl dane pracowników z departamentu 10 i 20, uporządkowane alfabetycznie według nazwisk (EMP, używając **in**)
- utworzyć raport:

  Who, what and when

  SMITH pracuje jako CLERK w oddziale 10 od 17-DEC-80

Rozwiązania:

wyświetl *ename*, *empno* i *sal* z tabeli 'EMP'

```
SQL> r
  1* select ename, empno, sal from emp where sal between 1000 and  2000

ENAME      EMPNO      SAL
---------- ---------- ----------
ALLEN      7499       1600
WARD       7521       1250
MARTIN     7654       1250
TURNER     7844       1500
ADAMS      7876       1100
MILLER     7934       1300

6 rows selected.
```

```
SQL> 2 order by sal;
SQL> l
  1  select ename, empno, sal from emp where sal between 1000 and  2000
  2* order by sal
SQL> r
  1  select ename, empno, sal from emp where sal between 1000 and  2000
  2* order by sal

ENAME      EMPNO      SAL
---------- ---------- ----------
ADAMS      7876       1100
WARD       7521       1250
MARTIN     7654       1250
MILLER     7934       1300
TURNER     7844       1500
ALLEN      7499       1600

6 rows selected.
```

wyświetl nazwy unikalnych stanowisk pracy uporządkowane alfabetycznie

```
SQL> select distinct job from emp order by job;

JOB
---------
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN


SQL> desc emp;

Name            Null?            Type
--------------- ---------------- ------------------
EMPNO           NOT NULL         NUMBER(4)
ENAME                            VARCHAR2(10)
JOB                              VARCHAR2(9)
MGR                              NUMBER(4)
HIREDATE                         DATE
SAL                              NUMBER(7,2)
COMM                             NUMBER(7,2)
DEPTNO          NOT NULL         NUMBER(2)
```

**wyświetl dane pracowników z departamentów 10 i 20,
uporządkowane alfabetycznie według nazwisk**

```
SQL> select ename, deptno, job from emp
  2  where deptno in (10,20)
  3  order by ename;

ENAME      DEPTNO     JOB
---------- ---------- --------
ADAMS      20         CLERK
CLARK      10         MANAGER
FORD       20         ANALYST
JONES      20         MANAGER
KING       10         PRESIDENT
MILLER     10         CLERK
SCOTT      20         ANALYST
SMITH      20         CLERK

8 rows selected.
```

uwórz raport postaci:

Who, what and when

SMITH pracuje jako CLERK w oddziale 10 od 17-DEC-80

```
SQL> r
  1  select initcap(ename) || ' pracuje jako ' ||
  2  initcap(job) || ' w oddziale ' || deptno ||
  3  ' od ' || hiredate
  4  "Who, what and when"
  5  from emp
  6* order by ename

Who, what and when
--------------------------------------------------------------------------------
Adams pracuje jako Clerk w oddziale 20 od 12-JAN-83
Allen pracuje jako Salesman w oddziale 30 od 20-FEB-81
Blake pracuje jako Manager w oddziale 30 od 01-MAY-81
Clark pracuje jako Manager w oddziale 10 od 09-JUN-81
Ford pracuje jako Analyst w oddziale 20 od 03-DEC-81
James pracuje jako Clerk w oddziale 30 od 03-DEC-81
Jones pracuje jako Manager w oddziale 20 od 02-APR-81
King pracuje jako President w oddziale 10 od 17-NOV-81
Martin pracuje jako Salesman w oddziale 30 od 28-SEP-81
Miller pracuje jako Clerk w oddziale 10 od 23-JAN-82
Scott pracuje jako Analyst w oddziale 20 od 09-DEC-82
Smith pracuje jako Clerk w oddziale 20 od 17-DEC-80
Turner pracuje jako Salesman w oddziale 30 od 08-SEP-81
Ward pracuje jako Salesman w oddziale 30 od 22-FEB-81

14 rows selected.
```

Wyrażenia w SQLu:

expr {= | != | ^= | _= | <\> | > | < | >= | <=} expr

(różnice: != *nie równy* dla VAX, UNIX, PC
^= *nie równy* dla IBM
_= *nie równy* dla IBM
<> *nie równy* - wszystkie systemy
)

{ expr [NOT] IN {expr_list | (subquery)}

expr [NOT] BETWEEN expr AND expr

expr IS [NOT] NULL

EXISTS (subquery)

{ (condition) | NOT condition | condition AND condition | condition OR condition }

Funkcje numeryczne, pojedyńcze wiersze[1]

Funkcja	Zwracana wartość
ABS(n)	Część całkowita *n*.
CEIL(n)	Najmniejsza liczba całkowita nie większa od *n.99*
COS(n)	Kosinus *n* (*n* w radianach).
COSH(n)	Kosinus hiperboliczny *n* (*n* w radianach).
EXP(n)	e^n.
FLOOR(n)	Najmniejsza liczba całkowita nie większa od *n*
LN(n)	Logarytm naturalny z *n*, (*n* > 0)
LOG(m,n)	$\log_m(n)$
MOD(m,n)	Reszta z dzielenia *m* przez *n*.
POWER(m,n)	m^n
ROUND(n[,m])	*n* zaokrąglone do *m* miejsc po przecinku (defaultowo *m* = 0).
SIGN(n)	**jeżeli** *n*<0 **to** -1; **jeżeli** *n*=0, **to** 0; **jeżeli** *n*>0, **to** 1.
SIN(n)	Sinus *n* (*n* w radianach)
SINH(n)	Sinus hiperboliczny *n* (*n* w radianach)
SQRT(n)	$\sqrt{n}$; **jeżeli** *n*<0, **to** *NULL*.
TAN(n)	Tangens *n* (*n* w radianach)
TANH(n)	Tangens hiperboliczny *n* (*n* w radianach)
TRUNC(n[,m])	*n* obcięte do *m* miejsc po przecinku (defaultowo *m* = 0).

[1] Wymienione zostały funkcje najczęściej używane - pełny zestaw funkcji zależy od konkretnej aplikacji.

Funkcje znakowe, pojedyńcze wiersze

Funkcja	Zwracana wartość
CHR(n)	zwraca znak o podanym kodzie
CONCAT($char_1$,$char_2$)	konkatenuje *$char_1$* i *$char_2$*.
INITCAP(char)	zmienia wszystkie pierwsze znaki w słowach na duże litery
LOWER(char)	zmienia wszystkie litery w *char* na małe
LPAD($char_1$,n[,$char_2$])	dostawia z lewej strony łańcucha *$char_1$ n* znaków *$char_2$*. Defaultowo *$char_2$* równy spacji.
LTRIM(char[,set])	usuwa z początku łańcucha *char* wszystkie znaki zgodne ze znakami z *set*. Defaultowo usuwa spacje.
REPLACE(char,search_string [,replacement_string])	w łańcuchu *char* zamień każde wystąpienie *search_string* na *replacement_string*. Jeżeli *replacement_string* nie występuje - usuń każde wystąpienie *search_string*.
RPAD($char_1$,n[,$char_2$])	dostawia z prawej strony łańcucha *$char_1$ n* znaków *$char_2$*. Defaultowo *$char_2$* równy spacji.
RTRIM(char[,set])	usuwa z końca łańcucha *char* wszystkie znaki zgodne ze znakami z *set*. Defaultowo usuwa spacje.
SOUNDEX(char)	zwraca kod 'słowa podobnego' do *char*. Nie działa dla polskich słowek.
SUBSTR(char,m[,n])	zwraca podłańcuch z *char*, zaczynający się od znaku o numerze *m* mający *n* znaków. Jeżeli *n* nie wystąpi - podłańcuch zaczynający się od znaku *m* zawierający wszystkie pozostałe znaki (do końca łańcucha *char*).
SUBSTRB(char,m[,n])	jak ***substr*** ale numery *m* oraz *n*okreœlaj[1] bajty, nie znaki (dla niektórych platform sprzêtowych to nie jest to samo)
TRANSLATE(char,from,to)	zamień w *char* wszystkie znaki występujące w łańcuhcu *from* na znali występujące w łańcuchu *to*
UPPER(char)	zmień wszystkie litery w *char* na duże litery
NLS_INITCAP(char [, nls_sort])	jak ***initcap*** ale pozwala zdefiniować '*narodowe´* zasady kapitalizacji wyrazów ('*nls_sort*')
NLS_LOWER(char [, nls_sort])	jak ***lower*** ale z możliwością zdefiniowania zasad '*narodowych*' zmiany liter na małe ('*nls_sort*')
NLS_UPPER(char [, nls_sort])	jak ***upper*** ale z możliwością zdefiniowania zasad '*narodowych*' zmiany liter na duże ('*nls_sort*')

Funkcje zwracające wyrażenia znakowe, działające na pojedyńczych wierszach

Funkcja	Zwracana wartość
ASCII(char)	zwraca kod ASCII znaku *char*. W systemach w których znaki mają kody wielobajtowe zwraca kod pierwszego bajtu.
INSTR($char_1$,$char_2$[,n[,m]])	zwraca pozycję m_{tego} (default: pierwszego) wystąpienia znaku $char_2$ w łańcuchu $char_1$. Zaczyna wyszukiwanie od n_{tego} (default: od pierwszego) znaku.
INSTRB(char1,char2[,n[,m]])	jak ***instr*** ale zwraca pozycję *bajtu* a nie znaku
LENGTH(char)	zwraca długość łańcucha *char* liczoną w znakach
LENGTHB(char)	zwraca długość łańcucha *char* liczoną w bajtach

Funkcje daty i czasu[2]

Funkcja	Zwracana wartość
ADD_MONTHS(d,n)	data *d* plus *n* miesiêcy
LAST_DAY(d)	data ostatniego dnai miesiąca do którego należy data *d*
MONTHS_BETWEEN(d,e)	ilość miesięcy o jaką data *e* poprzedza datę *d*
NEW_TIME(d,a,b)	jaka data w strefie *b* odpowiada dacie *d* w strefie *a*. *a* i *b* są znakowymi kodami stref czasowych
NEXT_DAY(d,char)	zwraca datę pierwszego dnia tygodnia o nazwie *char* która jest nie wcześniejsza niż data *d*
SYSDATE	bieżąca data i czas

'Inne' funkcje

Funkcja	Zwracana wartość
DECODE(expr,$search_1$,$return_1$, [$search_2$,$return_2$,]...[default])	**jeżeli** *expr* jest równe jakiejś wartości $search_i$ **to** zwróć odpowiadającą mu wartość $return_i$ **w przeciwnym razie** zwróć *default*
GREATEST(expr[,expr]...)	zwróć wartość maksymalną.
LEAST(expr[,expr]...)	zwróć wartość minimalną.
NVL($expr_1$,$expr_2$)	**jeżeli** $expr_1$ jest równe **null to** zwróæ wartoœæ $expr_2$ **w przeciwnym wypadku** zwróæ $expr_1$
UID	zwróć unikalny numer identyfikujący użytkownika
USER	nazwa bieżącego użytkownika
VSIZE(expr)	ilość bajtów zajmowanych przez wewnętrzną reprezentację *expr*
SYSDATE	bieżąca data i czas systemowy

[2]Na zmiennych typu **data** mo¿na wykonywaæ nastêpuj¹ce operacje:

data + liczba	oblicz datê o *liczba* dni póŸniejsz¹
data - liczba	oblicz datê o *liczba* dni wczeœniejsz¹
data - data	oblicz iloœæ dni pomiêdzy datami
data + liczba/24	dodaj pewn¹ iloœæ godzin do daty

Zaokrąglanie i 'obcinanie' daty i czasu

Funkcja	Zwracana wartość
ROUND(d[,fmt])	data *d* zaokrąglona według formatu *ftm*
TRUNC(d[,fmt])	data *d* 'obcięta' według formatu *ftm*

fmt:	cc	zamień na wiek
	y	zamień na rok (5)
	yy	zamień na rok (95)
	yyy	zamień na rok (995)
	yyyy	zamień na rok (1995)
	q	zamień na kwartał
	mm	zamień na miesiąc (1,..)
	mon	zamień na miesiąc (jan,...)
	month	zamień na miesiąc (january)
	ww	zamień na najbliższy poniedziałek (== początek tygodnia)
	w	zamień na najbliższy dzień który jest takim samym dniem tygodnia jak pierwszy dzień miesiąca wskazywanego przez datę
	j	zamień na numer dnia w tygodniu
	dd	zamień na numer dnia w miesiącu
	ddd	zamień na numer dnia w roku
	hh	zamień na godzinę
	hh12	zamień na godzinę w formacie am/pm
	hh24	zamień na godzinę w formacie 24 godzinnym
	mi	zamień na minutę

Funkcje konwersji

Funkcja	Zwracana wartość
TO_CHAR(expr[,fmt])	skonwertuj *expr* z wartości typu **number** lub **date** do wartości znakowej według formatu *fmt*.
TO_DATE(char[,fmt])	konweruje wyrażenie typu **char** do typu **date** według formatu *fmt*
TO_NUMBER(char [,fmt])	konwertuje wyrażenie typu **char** do typu **number** wed lug formatu *fmt*

Formaty konwersji, używane w funkcjach to_char *i* to_date

używane tylko w TO_CHAR

format	znaczenie
"string"	wypisywany bez zmian
fm	poprzedza **Month** lub **Day**, powoduje, że nazwy miesięcy i dni będą zajmowały tylko tyle znaków, ile występuje w nazwie.**w przeciwnym razie** wszystkie nazwy będą wyrównywane do tej samej długości
TH	dodane do numeru roku, miesiąca, dnia, godziny, minuty lub sekundy powoduje wypisanie przyrostka 'th', kapitalizacja liter - jak w formacie liczby
SP	dodane do liczby powoduje wypisanie jej wartości słownie
SPTH	jak **SP** + **TH**
THSP	jak **TH** + **SP**

N/w znaki będą ignorowane przez **TO_DATE** i wypisywane bez zmian przez **TO_CHAR**:

/	,	-	:
.	**A.M.**	**P.M.**	**a.m.**
p.m.	**AM**	**am**	**PM**
pm	**CC (wiek)**	**SCC (wiek, - dla pne)**	**B.C.**
BC	**AD**	**A.D.**	**bc**
b.c.	**ad**	**a.d.**	

wspólne dla TO_CHAR *i* TO_DATE

format	znaczenie
MM	numer miesiąca, cyframi arabskimi
RM	numer miesiąca, cyframi rzymskimi
MON	trzyliterowy skrót nazwy miesiąca, dużymi literami
Mon	j/w, ale tylko pierwsza litera duża
mon	jw, wszystkie litery małe
MONTH	pełna nazwa miesiąca, dużymi literami
Month	j/w, tylko pierwsza litera duża
month	j/w, wszystkie litery małe
DDD	numer dnia w roku
DD	numer dnia w miesiącu
D	numer dnia w tygodniu
DY	trójliterowy skrót nazwy dnia tygodnia, dużymi literami
Dy	j/w, tylko pierwsza litera duża
dy	j/w, wszystkie litery małe
DAY	pełna nazwa dnia tygodnia, wszystkie litery duże
Day	j/w, tylko pierwsza litera duża
day	j/w, wszystkie litery małe
YYYY	czterocyfrowy numer roku
SYYYY	j/w, dla lat pne - jako liczba ujemna
IYYY	j/w zgodny z normą ISO

format	znaczenie
YYY	trzy ostatnie cyfry numeru roku
IYY	j/w, ISO format
YY	dwie ostatnie cyfry roku
IY	j/w, ISO format
Y	ostatnia cyfra numeru roku
I	j/w, ISO format
RR	ostatnie dwie cyfry numeru roku, być może z innego wieku
YEAR	pełna, słowna nazwa roku, wszystkie litery duże
Year	j/w, tylko pierwsza litera duża
year	j/w, wszystkie litery małe
Q	numer kwartału w roku
WW	numer tygodnia w roku
W	numer tygodnia w miesiącu
IW	numer tygodnia w roku, ISO format
J	data Juliańska (od 31/12 4713 pne)
HH	numer godziny w dniu, 1 - 12
HH12	j/w
HH24	numer godziny, format 24 godzinny
MI	minuta w godzinie
SS	sekundy minuty
SSSSS	sekundy od północy

formaty używane do wypisywania liczb
***używane w* TO_CHAR*,* TO_NUMBER *i* COLUMN**

format	znaczenie
9	cyfra, ilość 9 określa szerokość pola
0	wyświetlaj wiodące zera
$	pozycja wyświetlania znaku $
.	pozycja kropki dzisiętnej
,	pozycja przecinka, oddzielającego trzycyfrowe grupy cyfr
MI	wypisuj minus **po prawej stronie** liczby ujemnej
PR	liczbe ujemne wyświetlaj w nawiasach
EEEE	wyświetlaj liczby w postaci wykładniczej
V	wyświetlaj liczbę pomnożoną przez 10^n, *n* jest ilością cyfr po **V**
B	wyświetlaj końcowe zera jako zera a nie spacje

Ćwiczenia

wyświetl wyśrodkowane nazwy departamentów, przyjmując, że nazwa departamentu ma 20 znaków

```
SQL> r
  1  select lpad(' ', (20-length(dname))/2) || dname
  2  "Departament"
  3* from dept

Departament
--------------------------------------------------------------------------------
    ACCOUNTING
     RESEARCH
      SALES
    OPERATIONS
```

wyświetl nazwiska pracowników oraz te same nazwiska z pierwszą literą 'L'zamienioną na 'X'

```
SQL> r
  1  select ename "Nazwisko",
  2  translate(substr(ename,1,instr(ename,'L')), 'L', 'X' )
  3  || substr( ename, instr(ename,'L')+1 )
  4  "po zamianie"
  5* from emp

Nazwisko   po zamianie
---------- --------------------
SMITH      SMITH
ALLEN      AXLEN
WARD       WARD
JONES      JONES
MARTIN     MARTIN
BLAKE      BXAKE
CLARK      CXARK
SCOTT      SCOTT
KING       KING
TURNER     TURNER
ADAMS      ADAMS
JAMES      JAMES
FORD       FORD
MILLER     MIXLER

14 rows selected.
```

wyświetl dane o pracownikach klasyfikując ich stanowiska pracy:
CLERK - urzędnik, MANAGER - szef, pozostali - pracownik

```
SQL> r
  1  select ename, job,
  2  decode( job, 'CLERK', 'urzednik', 'MANAGER', 'szef', 'pracownik' )
  3* from emp
```

ENAME	JOB	DECODE(JO
SMITH	CLERK	urzednik
ALLEN	SALESMAN	pracownik
WARD	SALESMAN	pracownik
JONES	MANAGER	szef
MARTIN	SALESMAN	pracownik
BLAKE	MANAGER	szef
CLARK	MANAGER	szef
SCOTT	ANALYST	pracownik
KING	PRESIDENT	pracownik
TURNER	SALESMAN	pracownik
ADAMS	CLERK	urzednik
JAMES	CLERK	urzednik
FORD	ANALYST	pracownik
MILLER	CLERK	urzednik

14 rows selected.

Funkcje grupowe

Funkcja	Zwracana wartość
AVG([DISTINCT|ALL]n)	wartość średnia *n*, wartości **null**s¹ ignorowane
COUNT([ALL]*)	ilość wierszy zwracana przez zapytanie lub podzapytanie, **'*'** oznacza '*policz ilość wierszy*'
COUNT([DISTINCT|ALL]expr})	ilość wierszy dla których *expr*ma wartoœæ ró¿n¹ od **null**
MAX([DISTINCT|ALL]expr)	maksymalna wartość *expr*
MIN([DISTINCT|ALL]expr)	minimalna wartość *expr*
STDDEV([DISTINCT|ALL]n)	odchylenie standardowe *n*, wartości **null** są ignorowane
SUM([DISTINCT|ALL]n)	suma wartości *n*
VARIANCE([DISTINCT|ALL]n)	wariancja *n*, wartości **null** są ignorowane

Wszystkie funkcje grupowe - oprócz **count(*)** - ignorują wartości **null**. Użycie klauzuli **group by** powoduje wykonywanie obliczeń grupowych na wydzielonych grupach rekordów. Jeżeli w klauzuli **group by** użyjemy nazw kilku kolumn otrzymamy 'grupy wewnątrz grup'. Kolejność grupowania - od lewej do prawej. **Na liście wyboru komendy *select* używającego funkcji grupowych wolno umieszczać tylko te kolumny, które występują w klauzuli *group by*** - oczywiście oprócz samych funkcji grupowych.

Można dodatkowo użyć klauzuli

having *wyrażenie logiczne*

Spowoduje to wybranie tylko tych grup, które spełniają warunek z **having**. Kolejność występowania klauzul w rozkazie **select** jest ściśle określona:

```
select      list kolumn
from        lista tablic
where       warunek dla wierszy
group by    kolumny
having      warunek dla grup
order by    kolumny;
```

wyświetl ilość pracowników z działu 20

```
SQL> select count(*)
  2  from emp
  3  where deptno=20;

COUNT(*)
----------
5
```

wyświetlić średnie zarobki na każdym stanowisku z wyjątkiem stanowiska 'MANAGER'

```
SQL> r
  1  select job, avg(sal)
  2  from emp
  3  where job <> 'MANAGER'
  4* group by job

JOB             AVG(SAL)
------          -------------
ANALYST         3000
CLERK           1037.5
PRESIDENT       5000
SALESMAN        1400
```

wyświetl średnie zarobki osiągane na różnych stanowiskach pracy w każdym departamencie

```
SQL> select deptno, job,avg(sal)
  2  from emp
  3  group by deptno, job;

DEPTNO    JOB             AVG(SAL)
-----------    ------          -------------
10        CLERK           1300
10        MANAGER         2450
10        PRESIDENT       5000
20        ANALYST         3000
20        CLERK           950
20        MANAGER         2975
30        CLERK           950
30        MANAGER         2850
30        SALESMAN        1400

9 rows selected.
```

Podzapytania

Sytuacja:

- znaleźć wszystkich pracowników o pensji równej minimalnej pensji w firmie

rozwiązanie:

- znajdź minimalną płącę

```
select          min(sal)
from            emp;
```

```
MIN(SAL)
------------
800
```

- znajdź pracowników o pensji równej wartości znalezionej w pierwszym wyszukiwaniu

```
select          ename, job, sal
from            emp
where           sal = 800;
```

```
ENAME      JOB          SAL
---------  ------       --------
SMITH      CLERK             800
```

można inaczej:

```
select          ename, job, sal
from            emp
where           sal = (  select   min(sal)
                         from     emp);
```

```
ENAME      JOB          SAL
---------  ------       --------
SMITH      CLERK             800
```

znajdź wszystkich pracowników zatrudnionych na tym samym stanowisku co BLAKE

```
select          ename, job
from            emp
where           job = (  select          job
                         from            emp
                         where           ename = 'BLAKE' );
```

```
ENAME      JOB
----------  --------
JONES      MANAGER
BLAKE      MANAGER
CLARK      MANAGER
```

to były zapytania zwracające pojedyńczą wartość, można również używać podzapytań zwracających grupę wartości:

- znajdź pracowników o najniższych pensjach w poszczególnych działach

```
select      ename, sal, deptno
from        emp
where       sal in ( select      min(sal)
                     from        emp
                     group by    deptno);
```

ENAME	SAL	DEPTNO
SMITH	800	20
JAMES	950	30
MILLER	1300	10

ponieważ wewnętrzne zapytanie używa **group by** zwróci więcej niż jedną wartość - dlatego musimy użyć **in** w warunku. Może się również zdażyć, że pracownik z działu ***x*** zostanie wyselekcjonowany ponieważ jego pensja będzie równa ***minimum z działu y***. Aby tego umiknąć możemy użyć *selectu* postaci:

```
select      ename, sal, deptno
from        emp
where       (sal,deptno)    in (    select      min(sal), deptno
                                    from        emp
                                    group by deptno);
```

ENAME	SAL	DEPTNO
SMITH	800	20
JAMES	950	30
MILLER	1300	10

konieczna zgodność kolejności, ilości i typów kolumn w **where** i podzapytaniu. Sygnalizowane są błędy jeżeli podzapytanie zwróci więcej niż jeden rekord a nie używamy operatorów '*grupowych*' oraz wtedy kiedy podzapytanie nie wybieże żadnego wiersza.

operatory *ANY*, *ALL* i *EXIST*

używane w podzapytaniach zwracających więcej niż jeden wiersz, w klauzulach **where** i **having** łącznie z operatorami porównywania (**=**, **!=**, **<**, **>**, **>=**, **<=)**

- znajdź pracowników zarabiających więcej niż najniższa pensja z wydziału 30

```
select      ename, sal, job, deptno
from        emp
where       sal >   any (   select      distinct sal
                            from        emp
                            where       deptno = 30)
order by    sal descending;
```

ENAME	SAL	JOB	DEPTNO
KING	5000	PRESIDENT	10
SCOTT	3000	ANALYST	20
FORD	3000	ANALYST	20
JONES	2975	MANAGER	20
BLAKE	2850	MANAGER	30
CLARK	2450	MANAGER	10
ALLEN	1600	SALESMAN	30
TURNER	1500	SALESMAN	30
MILLER	1300	CLERK	10
WARD	1250	SALESMAN	30
MARTIN	1250	SALESMAN	30
ADAMS	1100	CLERK	20

= any oznacza to samo co **in**. Zaleca się używanie **distinct** z **any** i **all** (zwiększa prędkość przetwarzania eliminując powtarzające się rekordy)

- znajdź pracowników których zarobki są wyższe od zarobków każdego pracownika z wydziału 30:

```
select      ename, sal, job, deptno
from        emp
where       sal >   all (   select      distinct sal
                            from        emp
                            where       deptno = 30)
order by    sal desc;
```

ENAME	SAL	JOB	DEPTNO
KING	5000	PRESIDENT	10
SCOTT	3000	ANALYST	20
FORD	3000	ANALYST	20
JONES	2975	MANAGER	20

podzapytań można również użyć w klauzuli **having** (używanej dla **group by**)

- znajdź departamenty w których średnia zarobków jest wyższa niż średnia zarobków w departamencie 30:

```
select      deptno, avg(sal)
from        emp
group by    deptno
having      avg(sal) > (    select      avg(sal)
                            from        emp
                            where       deptno = 30 );
```

```
    DEPTNO    AVG(SAL)
    ----------  ------------
            10     2916.67
            20        2175
```

- znajdź stanowisko pracy na którym są najwyższe średnie zarobki

```
select      job, avg(sal)
from        emp
group by    job
having      avg(sal) = (    select      max(avg(sal))
                            from        emp
                            group by    job);
```

```
    JOB             AVG(SAL)
    -----           ------------
    PRESIDENT            5000
```

nie można używać **order by** w podzapytaniu. Podzapytania mogą być dowolnie głęboko zagnieżdżone

- znajdź pracowników, których zarobki są wyższe niż maksymalna pensja z departamentu SALES:

```
select      ename, job, sal
from        emp
where       sal > (  select      max(sal)
                     from        emp
                     where       deptno = (    select  deptno
                                               from    dept
                                               where   dname = 'SALES'));
```

```
    ENAME   JOB             SAL
    --------  -----           -----
    JONES   MANAGER          2975
    SCOTT   ANALYST          3000
    KING    PRESIDENT        5000
    FORD    ANALYST          3000
```

podzapytania skorelowane

zwykle podzapytanie jest wykonywane tylko raz - służy do znalezienia wartosci, które będą używane do selekcji danych w zapytaniu zewnętrznym. Zapytanie skorelowane działa w/g schematu:

- pobierz wiersz poprzez zapytanie zewnętrzne
- wykonaj podzapytanie używając wartości z popranego (zewnętrznie) wiersza
- zaakceptuj lub odrzuć wiersz wybrany w zapytaniu zewnętrznym na podstawie podzapytania
- powtarzaj aż do wyczerpania wszystkich rekordów przez zapytanie zewnętrzne

- znajdź pracowników zarabiających więcej niż średnia zarobków w ich departamencie

```
select      ename, sal, deptno
from        emp e
where       sal > ( select      avg(sal)
                    from        emp
                    where       deptno = e.deptno )
order by    deptno;
```

ENAME	SAL	DEPTNO
KING	5000	10
JONES	2975	20
SCOTT	3000	20
FORD	3000	20
ALLEN	1600	30
BLAKE	2850	30

musimy użyć aliasu tablicy **emp** żeby móc skontruować warunek selekcji w skorelowanym podzapytaniu.

- znajdź pracowników, którzy posiadają podwładnych

```
select      ename, job, deptno
from        emp e
where       exists (    select      empno
                        from        emp
                        where       emp.mgr = e.empno )
order by    deptno;
```

ENAME	JOB	DEPTNO
CLARK	MANAGER	10
KING	PRESIDENT	10
JONES	MANAGER	20
SCOTT	ANALYST	20
FORD	ANALYST	20
BLAKE	MANAGER	30

- znajdź wydziały, w których nikt nie pracuje

```
select      deptno, dname
from        dept d
where       not exists (    select      1
                            from        emp e
                            where       e.deptno = d.deptno );
```

```
    DEPTNO     DNAME
----------     ---------
        40     OPERATIONS
```

podzapytanie nie musi zwracać wartości z tabeli, można użyć stałej dla zachowania poprawności składni.

ćwiczenia

- wyświetl nazwiska i zarobki trzech najlepiej zarabiających pracowników w firmie

```
select      ename, sal
from        emp e
where       3 > (   select      count(*)
                    from        emp
                    where       e.sal < sal );
```

```
ENAME     SAL
--------  -----
SCOTT      3000
KING       5000
FORD       3000
```

- wyświetl numer roku w którym zatrudniono najwięcej pracowników

```
select      to_char( hiredate, 'YYYY' ) year, count( empno ) number_of_emps
from        emp
group by    to_char( hiredate, 'YYYY' )
having      count( empno ) = (      select      max( count( empno ) )
                                    from        emp
                                    group by    to_char( hiredate, 'YYYY' ) );
```

```
      YEAR       NUMBER_OF_EMPS
      -------    ----------------------------
      1981                       10
```

Wybrane dane o użytkowniku

View	Zawartość
USER_CATALOG	tablice, widoki, synonimy i sekwencje należące do użytkownika
USER_COL_PRIVS	prawa dostępu do kolumn których użytkownik jest właścicielem, 'gwarantującym' lub 'posiadającym dostęp
USER_COL_PRIVS_MADE	wszystkie prawa dostępu do obiektów należących do użytkownika
USER_COL_PRIVS_RECD	prawa dostępu do kolumn dla których użytkownik jest 'gwarantującym'
USER_CONSTRAINTS	ograniczenia zdefiniowane dla tabel użytkownika-
USER_INDEXES	definicje indeksów użytkownika
USER_OBJECTS	obiekty należące do użytkownika
USER_SEQUENCES	sekwencje należące do użytkownika
USER_SNAPSHOTS	'snapshoty' które użytkownik może używać
USER_SOURCE	teksty opisujące obiekty użytkownika pamiętane w bazie
USER_SYNONYMS	synonimy zdefiniowane przez użytkownika
USER_SYS_PRIVS	przywileje systemowe zagwarantowane użytkownikowi
USER_TABLES	tablice należące do użytkownika
USER_TAB_COMMENTS	teksty komentarzy opisujących tabele i widoki użytkownika
USER_TRIGGERS	opisy triggerów użytkownika
USER_USERS	informacja o bieżącym użytkowniku
USER_VIEWS	definicje widoków należących do użytkownika

Wybrane dane o 'wszystkich:

View	Zawartość
ALL_CATALOG	wszystkie tabele, widoki, synonimy i sekwencje dostępne dla użytkownika
ALL_COL_PRIVS_MADE	prawa dostępu do kolumn dla których użytkownik lub PUBLIC jest gwarantującym dostęp
ALL_CONSTRAINTS	definicje ograniczeń dla tabel dostępnych dla użytkownika
ALL_ERRORS	informacje o błędach - dla wszystkich obiektów dostępnych użytkownikowi
ALL_INDEXES	definicje indeksów tabel dostępnych dla użytkownika
ALL_OBJECTS	obiekty dostępne dla użytkownika
ALL_SEQUENCES	definicje sekwencji distępnych dla użytkownika
ALL_SNAPSHOTS	'snapshoty' dostępne dla użytkownika
ALL_SOURCE	teksty definiujące obiekty dostępne dla użytkownika
ALL_SYNONYMS	synonimy dostępne dla użytkownika
ALL_TABLES	opisy tabel dostępnych dla użytkownika
ALL_TAB_COLUMNS	opisy kolumn z tabel i widoków dostępnych dla użytkownika
ALL_TAB_COMMENTS	teksty komentarzy dla tabel i widoków dostępnych dla użytkownika
ALL_TRIGGERS	triggery dostępne dla użytkownika
ALL_USERS	informacje o użytkownikach
ALL_VIEWS	tekstowe definicje widoków dostępnych dla użytkownika

Tworzenie tablic:

- **CREATE TABLE**

```
CREATE TABLE table
( { column [datatype] [column_constraint] ... | table_constraint}  ...
```

```
create table    dept
                (       deptno      number(2)     not null,
                        dname       char(12),
                        loc         char(12)
                );
```

można stworzyć tabelę poprzez podzapytanie

```
CREATE TABLE tablica
[ ( nazwa kolumny [ NULL / NOT NULL ] , ... ) ]
AS SELECT parametry select;
```

jeżeli nie podamy nazw kolumn - te które są wymienione w selekcie zostaną utworzone w tabeli. Jeżeli podamy nazwy kolumn - select zostanie użyty do wstawienia wartości do nowo utworzonej tabeli.

zasady ogólne:

- tworzenie nowych tabel i modyfikacja struktury istniejących tabel może się odbywać w czasie pracy ("*on-line*")

- postać nazwy:
  - do 30 znaków, musi się zaczynać od litery, może zawierać liery, cyfry, _, $, # ($ i # są 'nie zalecane')
  - musi być unikalna pośród nazw dostępnych dla danego użytkownika
  - jeżeli jest taka jak słowo kluczowe PL/SQL (nie zalecane) to musi być ujmowana w cudzysłowy

- jeżeli w instrukcji ***create table*** nazwę tabeli napiszemy w cudzysłowach (***"emp"***) to duże i małe litery w nazwach kolumn i nazwie tablicy nie są utożsamiane

Tworzenie tabel

SQL> ***select * from all_users;***

```
USERNAME        USER_ID         CREATED
------------    ----------      ---------------------
SYS             0               21-NOV-94
SYSTEM          5               21-NOV-94
SCOTT           8               21-NOV-94
SYSTEXT         23              27-JAN-95
SZCZYPKA        20              05-JAN-95
GASON           21              13-JAN-95
SDEMO           22              26-JAN-95
TEDD            18              05-JAN-95
KWIECIEN        19              05-JAN-95
T_USER          17              05-JAN-95
PINZUL          29              23-FEB-95
RURBANCZ        28              13-FEB-95

12 rows selected.
```

SQL> ***desc all_tables***;

```
Name                            Null?           Type
------------------------------  --------------  --------
OWNER                           NOT NULL        VARCHAR2(30)
TABLE_NAME                      NOT NULL        VARCHAR2(30)
TABLESPACE_NAME                 NOT NULL        VARCHAR2(30)
CLUSTER_NAME                                    VARCHAR2(30)
PCT_FREE                        NOT NULL        NUMBER
PCT_USED                        NOT NULL        NUMBER
INI_TRANS                       NOT NULL        NUMBER
MAX_TRANS                       NOT NULL        NUMBER
INITIAL_EXTENT                                  NUMBER
NEXT_EXTENT                                     NUMBER
MIN_EXTENTS                                     NUMBER
MAX_EXTENTS                                     NUMBER
PCT_INCREASE                                    NUMBER
BACKED_UP                                       VARCHAR2(1)
NUM_ROWS                                        NUMBER
BLOCKS                                          NUMBER
EMPTY_BLOCKS                                    NUMBER
AVG_SPACE                                       NUMBER
CHAIN_CNT                                       NUMBER
AVG_ROW_LEN                                     NUMBER
```

1* select owner, table_name from all_tables where owner='SCOTT'

```
OWNER       TABLE_NAME
------------    ----------------------
SCOTT       DEPT
```

SQL> ***connect scott/...***
Connected.

SQL> ***select table_name from user_tables;***

```
TABLE_NAME
------------------------------
BONUS
CUSTOMER
DEPT
DUMMY
EMP
IAPXTB
ITEM
ORD
PICTURES
PRICE
PRODUCT
SALGRADE

12 rows selected.
```

SQL> ***grant select on bonus to public;***

Grant succeeded.

SQL> ***l***
1* ***grant select on bonus to public***
SQL> ***c/bonus/customer/***
1* ***grant select on customer to public***
SQL> ***r***
1* ***grant select on customer to public***

Grant succeeded.

SQL> ***connect t_user/...***
Connected.
SQL> ***select table_name from all_tables where owner='SCOTT'***;

```
TABLE_NAME
------------------------------
BONUS
CUSTOMER
DEPT
EMP
ITEM
ORD
PICTURES
PRICE
PRODUCT

9 rows selected.
```

SQL> ***create table dept as select * from scott.dept***;

Table created.

SQL> ***select table_name from user_tables;***

```
TABLE_NAME
------------------------------
DEPT
T_TEST
```

SQL> ***desc dept***;

```
Name          Null?          Type
----------    -------------  -------------------
DEPTNO        NOT NULL       NUMBER(2)
DNAME                        VARCHAR2(14)
LOC                          VARCHAR2(13)
```

SQL> ***desc scott.dept;***

```
Name          Null?          Type
----------    -------------  -------------------
DEPTNO        NOT NULL       NUMBER(2)
DNAME                        VARCHAR2(14)
LOC                          VARCHAR2(13)
```

SQL> ***select * from dept;***

```
DEPTNO    DNAME           LOC
----------    ------------------    --------
10        ACCOUNTING      NEW YORK
20        RESEARCH        DALLAS
30        SALES           CHICAGO
40        OPERATIONS      BOSTON
```

SQL> ***select * from scott.dept;***

```
DEPTNO    DNAME           LOC
----------    ------------------    ---------
10        ACCOUNTING      NEW YORK
20        RESEARCH        DALLAS
30        SALES           CHICAGO
40        OPERATIONS      BOSTON
```

SQL> ***select table_name from all_tables where owner='SCOTT';***

```
TABLE_NAME
------------------------------
BONUS
CUSTOMER
DEPT
EMP
ITEM
ORD
PICTURES
PRICE
PRODUCT

9 rows selected.
```

SQL> ***create table emp as select * from scott.emp;***

Table created.

SQL> ***select * from emp;***

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	12501	400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	15000		30
7876	ADAMS	CLERK	7788	12-JAN-83	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

14 rows selected.

SQL> ***desc emp;***

Name	Null?	Type
EMPNO	NOT NULL	NUMBER(4)
ENAME		VARCHAR2(10)
JOB		VARCHAR2(9)
MGR		NUMBER(4)
HIREDATE		DATE
SAL		NUMBER(7,2)
COMM		NUMBER(7,2)
DEPTNO	NOT NULL	NUMBER(2)

Klauzule CONSTRAINTS

W instrukcjach *create table* i *alter table* można zdefiniować klauzule tworzące ograniczenia na dane wprowadzane do tablicy. Pozwala to na zdefiniowanie warunków - zarówno w obrębie jednej tablicy jak i wielu tablic (np. nie można zmienić numeru departamentu w którym pracuje pracownik - tablica EMP - o ile numer tego departamentu nie występuje w tablicy DEPT). *Oracle* zapewnia przestrzeganie tych ograniczeń podczas pracy z tabelemi (generuje *exceptions*).

Rodzaje ograniczeń:

- wartośc różna od NULL - klauzula **not null**

- występowanie tylko unikalnych wartości w kolumnie lub kolumnach - klauzula **unique**

  ... empno number(4) unique ...
  ... unique (empno, deptno) ...

- definiowanie kolumny lub kolumn jako kolumny ideksującej tablicę - klauzula **primary key**

  create table assignments
  (project number(4), employee number(4),
  primary key (project, employee));

  create table dept
  (deptno number primary key,
  ...);

  Tylko jedna klauzula ***primary*** może wystąpić w definicji tabeli. Żadna kolumna z atrybutem ***primary*** nie może mieć atrybutu ***null***.

- określenie związku '*klucz obcy*' - klauzula **foreign key**, może się odnosić zarówno do innej jak i do tej samej tabeli

  ... constraint emp_dept foreign key (mgr) references emp (empno) ...

  '*mgr*' jest '*kluczem obcym*' odnoszącym się do kolumny EMPNO tabeli EMP. '*emp_dept*' jest nazwą warunku *constraint*

- wymaganie aby wartości kloumny (kolumn) spełniały określony warunek - klauzula **check**

  ... hiredate date check (hiredate < sysdate) ...

```
create table    emp
               (empno     number(4) not null constraint emp_prim primary key,
                          ename          varchar(10) check (ename = upper(ename) ),
                          job            varchar(10),
column                    mgr            number(4),
constraints               hiredate       date check (hiredate < sysdate ),
                          ni_number      varchar(12),
                          sal            number(7,2),
                          deptno         number(2) not null
                                         constraint emp_dept references dept (deptno),
table                constraint emp_dept foreign key (mgr) references emp (empno)
constraints
               );
```

możliwe '*constraints*':

constraint *nazwa*	określa nazwę więzów. Jeżeli pominiemy nazwę - Oracle utworzy własną (*SYS_Cn*). Nazwa więzów pojawia się w komunikatach o błędach
null	defaultowo *not null*, określa czy kolumny muszą mieć nadane
not null	wartości
primary key (*kol*, …)	klucz tabeli; zawsze posiada własność *not null*
unique	wymaganie aby wszystkie wartości w kolumnie (we wszystkich rekordach) były wzajemnie różne. Kolumna powinna mieć atrybut *not null* i nie być *primary key*
foreign key (*kol*, ...)	ta kolumna będzie łączyć table (rózne lub rekordy tej samej
references	tabeli) poprzez wartości wskazanej kolumny. Kolumna ta musi
***tabela* (*kol*, ...)**	być typu *primary key* we wskazanej tablicy. '*references*' pozwoli na wstawienie wartości tylko wtedy, kedy występuje ona we wskazanej kolumnie wskazanej tabeli
check (*warunek*)	definiuje wrunek, który musi być spełniony w każdym wierszu tabeli. Może się odnosić tylko do wierszy tabeli w której jest definiowany

Uwaga: w definicjach więzów primary key i foreign key w wersji kolumnowej nie mogą pojawić się listy kolumn (*kol*, …)

Modyfikacja danych w tabeli:

- **INSERT**

```
INSERT INTO [schema.]{table | view}[@dblink]
        [ (column, ...) ]
        VALUES (expr, ...);

INSERT INTO [schema.]{table | view}[@dblink]
        [ (column, ...) ]
         subquery;
```

dodaje (append) nowy wiersz do tabeli

- można opuścić nazwy kolumn - o ile wstawiamy dane do wszystkich kolumn ale nie jest to zalecane - struktura tabeli może się zmienić (zostaną dodane nowe kolumny) i pojawią się błędy w aplikacjach
- wartości znakowe i daty muszą być ujęte w apostrofy. Jeżeli w dacie nie wstawiamy jawnie godziny to data jest wpisywana z czasem ustawionym na północ (00:00:00)
- użycie podzapytania umożliwia wstawienie wielu wierszy na raz. Nie wsytępuje wtedy słowo kluczowe *VALUES*.

- **DELETE**

```
DELETE [FROM] [schema.]{table | view}[@dblink] [alias]
        [WHERE condition];
```

usuwa z tabeli wiersze wybrane klauzulą *where*. **Jeżeli nie użyjemy *where* zostaną usunięte wszyskie wiersze tabeli!!!**

- **UPDATE**

```
UPDATE [schema.]{table | view}[@dblink] [alias]
        SET column = expr,
             column = (subquery), ...
        [WHERE condition]
```

zmienia wartości kolumn w wierszach wybranych klauzulą *where*. **Jeżeli nie użyjemy *where* zmodyfikowane zostaną wszystkie wiersze tabeli!** Można dowolnie mieszać użycie wyrażeń i podzapytań. Użycie konstrukcji

... (*kolumna*, *kolumna*, ...) = (*subquery*, *subquery*, ...) ...

pozwala nadać nazwy wielu kolumnom za pomocą jednego podzapytania

Modyfikacja struktury danych:

- **ALTER**

```
ALTER TABLE [schema.]table

    [ ADD (column [datatype] [DEFAULT expr] [column_constraint] )
      table_constraint
    ]

    [ MODIFY (column [datatype] [DEFAULT expr])
    ]

    [DROP INDEX index] ... [DROP TRIGGER trigger] ...
    [DROP SYNONYM synonim] ... [DROP VIEW widok] ...
```

- ***add*** służy do dodawania nowych kolumn lub więzów (*constraints*)

```
alter table  emp
add          (spouse_name cher(10));

alter table  emp
add          (check (sal <= 5000 ));
```

- ***modify*** służy do zmodyfikowania definicji istniejącej kolumny

```
alter table  emp
modify               (ename char(25));
```

ograniczenia:

- nie można zmieniać atrybutu na *not null* jeżeli w kolumnie występują wiersze z wartością *null*
- nie można dodać nowej kolumny typu *not null*. Trzeba dodać kolumnę jako *null*, a następnie zmienić jej typ na *not null*.
- nie można zmniejszyć rozmiaru kolumny – chyba, że jest pusta
- nie można skasować kolumny (w Oracle 10g już można, np.: **alter table emp drop column comm;**)
- nie można zmienić nazwy kolumny (w Oracle 10g już można, np.: **alter table emp rename column job to position;**)

- **DROP**

DROP TABLE [schema.]table

[CASCADE CONSTRAINTS] ;

usuwa wszystkie dane i wartości indeksów. **Operacja nieodwracalna!!!**. Klauzula '*cascade constraints*' powoduje usunięcie wszystkich więzów związanych z usuwaną tablicą.

Usuwane są dane i definicja tabeli.

- **TRUNCATE *(tylko w Oracle 7)***

TRUNCATE TABLE [schema.]table ;

usuwa wszystkie dane z tablei, pozostawiając jej definicję (tabela jest pusta). **Operacja nieodwracalna!!!**

pomocnicze

- **COMMENT ON TABLE *nazwa* IS *'tekst'*;**

definiuje komentarz do tabeli

- **COMMENT ON COLUMN *nazwa_tabeli.nazwa_kolumny* IS *'tekst'*;**

definiuje komentarz do kolumny

usunięcie - przez ponowne zdefiniowanie z pustym tekstem(''). Wartości komentarzy znajdują się w tabelach ALL_COL_COMMANTS i USER_COL_COMMENTS, kolumna COMMENTS.

- **RENAME *stara_nazwa* TO *nowa_nazwa*;**

zmienia nazwę tabeli. **Trzeba samemu zmienić odwołania do tabeli we wszystkich operacjach!!!**

Zaawansowane postacie instrukcji *select*

Łączenie tablic

- działają na wynikach kilku niezależnych zapytań, umożliwiają stworzenie sumy, przecięcia i różnicy zapytań

suma ("*or*")
wszystkie - wzajemnie różne - wiersze zwracane w wyniku obu rozkazów *select*

```
select      job
from        emp
where       deptno = 10
union
select      job
from        emp
where       deptno = 30;
```

```
JOB
---------
CLERK
MANAGER
PRESIDENT
SALESMAN
```

przecięcie ("*and*")
wiersze wspólne - zwracane w wyniku każdego rozkazu *select*

```
select      job
from        emp
where       deptno = 10
intersect
select      job
from        emp
where       deptno = 30;
```

```
JOB
---------
CLERK
MANAGER
```

różnica ("*minus*")
zwracane przez pierwszy *select* i nie zwracane przez drugi

```
select      job
from        emp
where       deptno = 10
minus
select      job
from        emp
where       deptno = 30;
```

```
JOB
---------
PRESIDENT
```

Można używać więcej niż dwóch *select*ów połączonych operatorami działającymi na zbiorach (set). Kolejność - od góry do dołu, modyfikowana nawiasami. W zapytaniu z operatorami na zbiorach **order by** występuje tylko raz - *na końcu ostatniego select*. W tym **order by** nie używamy nazwy kolumny tylko numeru na liście pierwszego *select*u (tak zawsze można)

```
select      empno, ename, sal
from        emp
union
select      id, name, salary
from        emp_history
order by 2;
```

ograniczenia

- każdy *select* musi mieć tę samą ilośc kolumn, tego samego typu
- powtarzające się wiersze są standrdowo eliminowane (nie można użyć *distinct*) chyba że użyjemy **all**
- nagłówki kolumn są tworzone na podstawie pierwszego *select*u
- **order by** występuje po ostatnim *select*, w **order by** używamy numerów a nie nazw kolumn
- operatory działające na zbiorach mogą występować w podzapytaniach
- kolejnośc - od góry do dołu, może być zmieniana nawiasami

ćwiczenie:

- znajdź stanowisko (job) do którego przyjmowano pracowników zarówno w pierwszej połowie 1983 roku jak i w pierwszej połowie 1984 roku

```
select      job
from        emp
where       hiredate between '01-JAN-83' and '30-JUN-83'
intersect
select      job
from        emp
where       hiredate between '01-JAN-84' and '30-JUN-84'
```

```
JOB
------
MANAGER
SALESMAN
```

Wybieranie danych z kilku tablic

Używamy normalnej instrukcji **select**, wymieniając w **from** kilka tablic. Możemy po nazwach tablic podać aliasy (do 30 znaków) oraz użyć - tych aliasów lub wprost nazw tablic - do klasyfikowania nazw kolumn:

```
select ename, job, dname
from emp, dept
where emp.deptno = dept.deptno;

select e.ename, job, dname
from emp e, dept
where e.deptno = dept.deptno;
```

Użycie aliasów lub nazw tablic do kwalifikowania odwołania do kolumny przyspiesza działanie **select** i usuwa ewentualne niejednoznaczności. Jeżeli zdefiniujemy alias dla tablicy to nie możemy używać jej nazwy do kwalifikowania kolumn - trzeba użyć aliasu.

Jeżeli nie użyjemy '**where**' to dostaniemy *iloczyn kartezjański dwóch tablic* - zwykle jest to zbyt wiele wierszy by można je było sensownie użyć. W warunku **where** może wystąpić dowolne wyrażenie logiczne - można używać **between**, **like**, etc.

Warunek z **where** decyduje, które wiersze zostaną wybrane do przetwarzania. Te, które nie spełniaja warunku z **where** zostaną odrzucone. Jeżeli chcemy żeby z pewnej tablicy wybrać wszystkie rekordy i połączyć je z wyselekcjowanymi elementami innej tablicy - umieszczamy w warunku **where** po tej stronie wyrażenia gdzie znajduje się tablica z której chcemy wybrać wszystkie elementy - plus w nawaisach ('**(+)**')

```
select e.ename, d.deptno, d.name
from emp e, deptno d
where e.deptno (+) = d.deptno and d.deptno in (30,40);
```

Można użyć tylko jednego '(+)' w całym selekcie. Można powiązać tabelę z samą sobą - tworząc warunek w **where** z użyciem kolumn z tej samej tablicy.

wybierz nazwiska wszystkich pracowników wraz z numerami i nazwami departamentów , w których są zatrudnieni

```
select      e.ename, d.deptno, d.name
from        emp e, deprno d
where       e.deptno(+) = d.deptno and d.deptno in (30,40);
```

ENAME	DEPTNO	DNAME
CLARK	10	ACCOUNTING
KING	10	ACCOUNTING
MILLER	10	ACCOUNTING
SMITH	20	RESEARCH
ADAMS	20	RESEARCH
FORD	20	RESEARCH
SCOTT	20	RESEARCH
JONES	20	RESEARCH
ALLEN	30	SALES
BLAKE	30	SALES
MARTIN	30	SALES
JAMES	30	SALES
TURNER	30	SALES
WARD	30	SALES

14 rows selected.

utwórz listę pracowników (nazwisko, stanowisko, zarobki) wraz z stawką zaszeregowania (tabela *salgrade*)

```
select      ename, job, sal, grade
from        emp, salgrade
where       sal bettwen losal and hisal;
```

ENAME	JOB	SAL	GRADE
SMITH	CLERK	800	1
ADAMS	CLERK	1100	1
JAMES	CLERK	950	1
WARD	SALESMAN	1250	2
MARTIN	SALESMAN	1250	2
MILLER	CLERK	1300	2
ALLEN	SALESMAN	1600	3
TURNER	SALESMAN	1500	3
JONES	MANAGER	2975	4
BLAKE	MANAGER	2850	4
CLARK	MANAGER	2450	4
SCOTT	ANALYST	3000	4
FORD	ANALYST	3000	4
KING	PRESIDENT	5000	5

14 rows selected.

wyświetl dane o pracownikach osiągających roczne zarobki w wysokości 36000 oraz o tych, którzy są urzędnikami (stanowisko 'CLERK)

wyświetlaj: nazwisko, stanowisko, roczne zarobki, numer i nazwę departamentu oraz grupę zaszergowania

```
select      ename,
            job,
            sal * 12 + nvl(comm,0) ANNUAL_SAL,
            d.deptno,
            dname,
            grade
from        emp e, salgrade, dept d
where       e.deptno = d.deptno
      and   sal between losal and hisal
      and   ( sal*12+nvl(comm,0) = 36000 or e.job = 'CLERK' )
order by    e.job;
```

ENAME	JOB	ANNUAL_SAL	DEPTNO	DNAME	GRADE
SCOTT	ANALYST	36000	20	RESEARCH	4
FORD	ANALYST	36000	20	RESEARCH	4
MILLER	CLERK	15600	10	ACCOUNTING	2
JAMES	CLERK	11400	30	SALES	1
ADAMS	CLERK	13200	20	RESEARCH	1
SMITH	CLERK	9600	20	RESEARCH	1

6 rows selected.

wyświetlić nazwiska i zarobki tych pracowników, którzy zarabiają mniej od swoich kierowników

```
SQL> select e.ename emp_name, e.sal emp_sal, m.ename mgr_name, m.sal mgr_sal
  2  from emp e, emp m
  3  where e.mgr = m.empno and ( e.sal < m.sal );
```

EMP_NAME	EMP_SAL	MGR_NAME	MGR_SAL
ALLEN	1600	BLAKE	2850
WARD	1250	BLAKE	2850
JAMES	950	BLAKE	2850
TURNER	1500	BLAKE	2850
MARTIN	1250	BLAKE	2850
MILLER	1300	CLARK	2450
ADAMS	1100	SCOTT	3000
JONES	2975	KING	5000
CLARK	2450	KING	5000
BLAKE	2850	KING	5000
SMITH	800	FORD	3000

11 rows selected.

znajdż departament w którym nikt nie pracuje

```
select      deptno, dname
from        dept
minus
select      emp.deptno, dname
from        emp, dept
where       emp.deptno = dept.deptno;
```

wyświetlić listę pracowników zaznaczając gwiazdką ostatnio zatrudnionego

ENAME	HIREDATE	MAXDATE
ADAMS	12-JAN-83	*
ALLEN	20-FEB-81	
BLAKE	01-MAY-81	
CLARK	09-JUN-81	
FORD	03-DEC-81	
JAMES	03-DEC-81	
JONES	02-APR-81	
KING	17-NOV-81	
MARTIN	28-SEP-81	
MILLER	23-JAN-82	
SCOTT	09-DEC-82	
SMITH	17-DEC-80	
TURNER	08-SEP-81	
WARD	22-FEB-81	

14 rows selected

```
select    ename, hiredate, '*' MAXDATE
from      emp
where     hiredate = (  select  max(hiredate)
                        from    emp )
union
select    ename, hiredate, ' '
from      emp
where     hiredate != (  select  max(hiredate)
                         from    emp );
```

PL/SQL - zmienne, podstawianie wartości

- konstrukcja **&zmienna** umieszczona w pliku komend PL/SQLa powoduje zapytanie użytkownika o wartość tej zmiennej ('*Enter value for nazwa_zmiennej:*'). Wartość ta musi być wprowadzana każdorazowo podczas uruchamiania pliku z komendami PL/SQL - wartości te nie są pamiętane.

  Wartości znakow i daty należy wprowadzać w apostrofach. Aby uniknąć wpisywania apostrofów można użyć konstrukcji:

```
select ename, deptno, &expression
from emp
where job='&job_title';
```

  za pomocą konstrukcji z '&' możemy wprowadzać wartości proste albo wyrażenia - po pytaniu o wartość zmiennej można wpisać wyrażenie pod warunkiem że jest ono w tym momencie w pełni obliczalne

  Jeżeli nie chcemy wprowadzać wartości każdorazowo - użyć konstrukcji **&&zmienna** - pierwsze wykonanie komend PL/SQL spowoduje wczytanie zmiennej, w następnych będzie używana zapamiętana wartość

- **DEFINE** - do definiowania zmiennych i wyświetlana ich wartości. Użycie komendy **DEFINE *nazwa_zmiennej*** spowoduje wyświetlenie wartości tej zmiennej lub napisu **undefined**

  Konstrukcja

  define zmienna = wyrażenie

  powoduje zdefiniowanie zmiennej 'globalnej dla sesji'. Tak zdefiniowaną zmienną możemy użyć poprzez **&zmienna** (pojedyńczy apostrof). Usuwamy definicje komendą

  undef zmienna
  undefine zmienna

- **parametry wywołania komend SQL - poprzez *start***

  Można wywołać komendy PL/SQLa przekazując do nich parametry:

  start plik par$_1$ par$_2$ ... par$_9$

  W pliku z komendami odwołujemy się do parametrów poprzez **&1, &2, ... &9** (pojedyńczy &).Tylko komenda **start** pozwala na przekazanie parametrów - **run** nie może mieć parametrów.

- komenda **ACCEPT**

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

  powoduje wypisanie promptu oraz wczytanie wartości do zmiennej. O ile wystąpi **hide** wprowadzana wartość nie jest wyświetlana. **noprompt** powoduje tylko ustawienie kursora w nowej linii bez wypisywania tekstu. Jeżeli podaliśmy typ zmiennej badana będzie poprawność wprowadzonej wartości. Do zmiennej typu **char** można wprowadzać teksty i liczby, do zmiennej typu **number** tylko liczby.

zastosowanie:

```
select      empno, ename, sal
from        emp
where       deptno = &departament_numb;
```

```
select      ename, deptno, sal*12
from        emp
where       job = '&job_name';
```

```
select      empno, ename, sal
from        emp
where       deptno = &&departament_numb;
```

```
define      rem = 'sal * 12 + nvl( comm,0 )

select      ename, job, &rem
from emp
order by    &rem

undefine    rem
```

```
select      empno, ename, sal
from        emp
where       job = '&1';
```

```
save job1
```

```
start job1 clerk
```

EMPNO	ENAME	SAL
7369	SMITH	800
7876	ADAMS	1100
7900	JAMES	950

SELECT - pełna postać:

```
SELECT [DISTINCT | ALL] { *
        | { [schema.]{table | view | snapshot}.*
        | expr [ [AS] c_alias ] }
        [, { [schema.]{table | view | snapshot}.*
        | expr [ [AS] c_alias ] } ] ... }

FROM { (subquery)
        | [schema.]{table | view | snapshot}[@dblink] } [t_alias]
        [, { (subquery)
        | [schema.]{table | view | snapshot}[@dblink] }
[t_alias] ] ...

[WHERE condition ]

[ [START WITH condition] CONNECT BY condition]

[GROUP BY expr [, expr] ...] [HAVING condition]

[{UNION | UNION ALL | INTERSECT | MINUS} SELECT command ]

[ORDER BY {expr | c_alias | position} [ASC | DESC]
        [, {expr | c_alias | position}
        [ASC | DESC]] ...]

[FOR UPDATE [OF [[schema.]{table | view}.]column
        [, [[schema.]{table | view}.]column] ...]
         [NOWAIT] ]
```

- sterowanie sesją (wybrane parametry sesji- '*zmienne środowiskowe*')

echo off **echo on**	włącz / wyłącz wyświetlanie wykonywanych instrukcji
feed[back] n **feed[back] off** **feed[back] on**	'*n*' - jeżeli *select* zwrócił nie mniej niż *n* wierszy - wyświetl komunikat. *on/off* - włącz lub wyłącz to wyświetlanie, default: *feedback 6*
lin[esize] n	szerokośc linii, nie więcej niż 500 znaków, default: 80
heading off **heading on**	wyświetlaj / nie wyświetlaj nagłówki kolumn
numf[ormat] tekst	definiuje format wyświetlania liczb, znaczenie *tekst* - zob. dalej instrukcję **column**
num[width] n	definiuje szerokość wyświetlania liczb
pages[ize] n	ilość linii na stronie, redefiniuje ilośc linii po których wyświetlanie zatrzymuje się czekając na naciśnięcie dowolnego klawisza
pause off **pause on** **pause tekst**	**on** - czekaj na naciśnięcie klawisza po kadej stronie, **off** - nie czekaj; **tekst** - wyświetlaj ten tekst zamiast defaultowego napisu
ver[ify] off **ver[ify] on**	**on** powoduje wyświetlanie rozkazu PL/SQL przed i po podstawieniu wartości pod '&nazwa' i '&&nazwa'
timi[ng] off **timi[ng] on**	wyświetlaj informacje o czasie wykonania każdej instrukcji SQL
spa[ce] n	definiuje odstęp między kolumnami w czasie wyświetlania; default: 1

bieżace wartości można wyświetlać komendą **show nazwa_zmiennej**. **show all** wyświetla wartości wszystkich zmiennych. Ustawianie wartości zmiennych środowiskowych można wpisać do pliku ***login.sql***.

polecenie **column**:

col[umn] nazwa_kolumny lista_opcjii
col[umn] alias_kolumny lista_opcjii

muszą być wpisane do pliku **.sql** i wykonane przez ***start nazwa_pliku parametry*** lub wpisywane z linii komend PL/SQLa - są wtedy natchmiast wykonywane, nie niszczą zawartości bufora komend SQLa

parametry:

format opis	definiuje sposób wyświetlania kolumny, możliwe wartości: An pole znakowe o szerokości *n* 9 pozycja na cyfrę 0 wyświetlaj wiodące zera $ wyświetlaj **$** . kropka dziesiętna , separator tysięcznych MI minus po lewej EEEE notacja wykładnicza B wartość równa 0 nie będzie wyświetlana
wrap	przenoś tekst do następnej linii jeżeli nie mieści się w kolumnie (default)
trunc	obcinaj napis jeżeli nie mieści się w kolumnie
word_wrapped	przenoś całe słowa (nie dizel słów) chyba że słowo nie mieści się w kolumnie
clear	usuń zdefiniowane wcześniej formatowania dla tej kolumny
heading tekst	tworzy nagłowki wyselekcjonowanych kolumn, | w tekście powoduje łamanie linii
justify left **justify right** **justify center**	sposób justowania nagłówków kolumn, default: **char** / **date** - wyrównane do lewej **number** - wyrównane do prawej
like **nazwa_kolumny**	kopiuje formatowanie ze wskazanej kolumny
null tekst	***tekst*** będzie wyświetlany zamianst **null** w kolumnach

pozostałe polecenia:

ttitle 'ciąg_znaków'	tworzy nagłówek każdej strony: wyświetla datę w lewym górnym rogu strony oraz **ciąg_znaków** z prawej
ttitle	wyświetl bieżące ustawienie ttitle
ttitle off	nie wyświetlaj
btitle 'ciąg_znaków'	definiuje stopkę - **ciąg_znaków** będzie centrowany
btitle	wyświetl bieżące ustawienie btitle
btitle off	nie wyświetlaj

dodatkowe parametry *ttitle* i *btitle*:

col n	ustaw się w kolumnie *n*
skip n	*n* razy przejdź do nowej linii
tab n	skocz o *n* punktów tabulacji
left **center** **right**	sposób justyfikacji tekstu z **ttitle** i **btitle**
format opis	sposób formatowania tekstu, znaczenie pol - zob. wyżej

zmienne *'systemowe'*:

SQL.PNO	numer bieżącej strony raportu
SQL.LNO	numer bieżącej linii
SQL.USER	bieżąca nazwa użytkownika *Oracla*
SQL.SQLCODE	kod błędu ostatnio wykonywanego rozkazu SQL

instrukcja *break*

postać:

break on *kolumna akcja*

tylko jedno polecenie jest bieżaco aktywne. Instrukcja powoduje wykonanie określonej akcji jeżeli zmieni się wartość wskazanej kolumny. Dlatego raport powinien być generowany z klauzulą **order by** - inaczej dostaniemy prawie co chwilę działanie **break**. Jeżeli używaliśmy instrukcji ***col*** do wyświetlania kolumn w instrukcji select to w ***break*** powinniśmy użyć nazwy występującej w ***col*** zamiast nazwy kolumny.

akcje:

page skocz do nowej strony
skip n *n* razy przejdź do nowej linii

kasowanie instrukcji **break**:

clear breaks

instrukcja *'compute'*

postać:

compute ***lista_funkcji_grupowych*** **of** ***lista_kolumn*** **on** ***przerwania***

powoduje wykonanie obliczeń - wymienionymi funkcjami grupowymi - na wskazanych kolumnach. *'przerwania'* to kolumny użyte w **order by** lub słówko **report** - oznacza zakres całego raportu. Podobnie jak w ***break*** - jeżeli uzywamy instrukcji ***col*** w ***select*** to w instrukcji ***compute*** używamy nazw występujących w ***col*** zamiast nazw kolumn. Instrukcja ***compute*** działa tylko dla tych kolumn, dla których zdefiniowaliśmy akcje w instrukcji ***break***.

funkcje grupowe:

avg	średnia
cou[nt]	ilość wystąpień nie pustych wartości
max[imum]	wart. max.
min[imum]	wart. min.
num[ber]	ilość wierszy
std	standardowe odchylenie
sum	suma niepustych wartości
var[iance]	wariancja

przykład:

compute sum avg of sal comm on deptno report;

sumuj pensje (**sal**) , obliczaj wartość średnią **comm** w każdej sekcji raportu opartej na kolumnie **deptno** oraz na końcu całego raportu.

ćwiczenie:

utwórz raport:

Thu Nov 11 **page 1**

EMPLOYEE REPORT

Departament	Job	Emp. No.	Name	Hire Date	Monthly Salary	Annual Comm	Total
ACCOUNTING	CLERK	7934	MILLER	01/82	1,300.00		15,600.00
ACCOUNTING	MANAGER	7782	CLARK	06/81	2,450.00		29,400.00
ACCOUNTING	PRESIDENT	7839	KING	11/81	5,000.00		60,000.00
RESEARCH	ANALYST	7902	FORD	12/81	3,000.00		36,000.00
RESEARCH	ANALYST	7902	FORD	12/81	3,000.00		36,000.00
RESEARCH	ANALYST	7788	SCOTT	12/82	3,000.00		36,000.00
RESEARCH	CLERK	7369	SMITH	12/80	800.00		9,600.00
RESEARCH	CLERK	7876	ADAMS	01/83	1,100.00		13,200.00
RESEARCH	MANAGER	7566	JONES	04/81	2,970.00		35,700.00
SALES	CLERK	7900	JAMES	12/81	950.00		11,400.00
SALES	MANAGER	7698	BLAKE	05/81	2,850.00		34,200.00
SALES	SALESMAN	7499	ALLEN	02/81	1,600.00	300.00	19,500.00
SALES	SALESMAN	7654	MARTIN	09/81	1,250.00	1,400.00	16,400.00
SALES	SALESMAN	7844	TURNER	09/81	1,500.00	0.00	18,000.00
SALES	SALESMAN	7521	WARD	02/81	1,250.00	500.00	15,500.00

CONFIDENTIAL

```
set echo off
set pagesize 24
set feedback off
set linesize 78

col a     format a10            heading 'Departament'
col b     format a9             heading 'Job'
col c     format 9999           heading 'Emp.|No.'
col d     format a8             heading 'Name'
col e     format a5             heading 'Hire|Date'
col f     format b99,999.99     heading 'Monthly|Salary'
col g     format 9,999.99       heading 'Annual|Comm'
col h     format 999,999.99     heading 'Total'

ttitle    'EMPLLOYEE REPORT'
btitle    'CONFIDENTIAL'

select          dname a,
                job b,
                empno c,
                ename d,
                to_char( hiredate, 'MM/YY' ) e,
                sal f,
                comm g,
                sal * 12 + nvl( comm,0 ) h
from      emp, dept
where     emp.deptno = dept.deptno
order     by dname, job;

clear           columns
ttile           off
btitle          off
set             feedback on
set             pages 24
```

'Drzewa zależności'

- jeżeli tablica zawiera wewnętrze powiązania elementów to można wygenerować raport typu 'drzewo powiązań':

```
break on      deptno skip 1
select        level, deptno, empno, ename, job, sal
from          emp
connect by    prior empno = mgr
start with    mgr is null;
```

LEVEL	DEPTNO	EMPNO	ENAME	JOB	SAL
1	10	7839	KING	PRESIDENT	5,000
2	20	7566	JONES	MANAGER	2,975
3		7788	SCOTT	ANALYST	3,000
4		7876	ADAMS	CLERK	1,100
3		7902	FORD	ANALYST	3,000
4		7369	SMITH	CLERK	800
2	30	7698	BLAKE	MANAGER	2,850
3		7499	ALLEN	SALESMAN	1,600
3		7521	ARD	SALESMAN	1,250
3		7654	MARTIN	SALESMAN	1,250
3		7844	TURNER	SALESMAN	1,500
3		7900	JAMES	SALESMAN	950
2	10	7782	CLARK	MANAGER	2,450
3		7934	MILLER	CLERK	1,300

uwagi:

connect by	definiuje strukturę powiązań, musi wystąpić
prior	definiuje 'podrzędność' i 'nadrzędność' rekordów powiązanych klauzulą *connect by*; działanie: znajdź wartość dla pola z *prior* a następnie wszystkie rekordy powiązane z tym rekordem
start with	opcjonalna, definiuje warunek początku przeszukiwania

- ten typ wyszukiwania działa tylko w obrębie pojedyńczej tabeli
- jeżeli rekord nie ma powiązania w/g zdefiniowanej relacji to nie jest selekcjonowany
- można użyć *order by* (musi wtedy wystąpić jako ostatnia klauzula *select*) ale nie jest to zalecane - może zaburzyć wyświetlaną strukturę dzewa powiązań
- za pomocą *where* i *connect by* można wybierać gałęzie które będą wyświetlane: wyeliminowanie przez *where* usuwa pojedyńczy rekord, przez *connect by* - całą gałąź

Transakcje

transakcja to operacje na bazie danych złożona z szergu zmian w jednej lub wielu tabelach, traktowana przez system jako zamknięta całość. Transakcja (jej zakres) jest definiowana jednym z niżej wymienionych warunków:

- zaczyna się pierwszym rozkazem DML[3] lub DDL[4]
- kończy się wystąpieniem jednego z rozkazów:
  - COMMIT / ROLLBACK
  - rozkaz DDL
  - niektóre błędy ('*deadlock*') związane z blokowaniem rekordów
  - koniec sesji ('*exit*')
  - awaria komputera

transakcja jest zatwierdzana tylko wtedy kiedy cała wykona się poprawnie. W przeciwnym wypadku stan bazy jest odtwarzany.

- **COMMIT**

COMMIT [WORK];

zatwierdza i kończy bieżącą transakcję. '*work*' jest opcjonalne - nie ma żadnego znaczenia. Każdy rozkaz DDL powoduje wykonanie *commit* w trakcie jego wykonania.

- **SAVEPOINT**

SAVEPOINT *nazwa*

definiuje 'punkt zachowania' - pozwala podzielić transakcje na mniejsze części. Można mieć do 5 punktów zachowania (to stała dla użytkownika, modyfikowalna na poziomie zarządzania Oraclem). Ponwne zdefiniowanie punktu zachowania o tej samej nazwie usuwa starą definicję. Punkty zachowania są zorganizowane stosowo.

- **ROLLBACK**

ROLLBACK [WORK]
[TO [SAVEPOINT] *savepoint*];

powoduje odtworzenie stanu bazy - albo do początku transakcji albo do stanu definiowanego punktem zachowania.

- jeżeli podamy nazwę punktu zachowania to zachowujemy ten punkt zachowania lecz tracimy wszystkie punkty zachowanie zdefiniowane 'później'.
- *rollback* bez nazwy punktu zachowania zamyka bieżącą transakcję i kasuje wszelkie punkty zachowania

[3] **D**ata **M**anipulation **L**anguage - instrukcje zmieniaj¹ce **dane** w tabelach
[4] **D**ata **D**efinition **L**anguage - instrukcje zmieniaj¹ce **strukturê** tabeli

```
insert into  dept
values       (50, 'Testing', 'Las vegas');

savepoint   insert_done;

update       dept
set          dname = 'MARTETING';

rollback to insert_done;

update       dept
set          dname = 'MARKETING'
where        dname = 'SALES';

commit;
```

Automatyczny *COMMIT*

można ustawić SQLPLUS w tryb pracy '*auto commit*':

set auto[commit] on	powoduje automatyczne wykonywane *commit* po każdym *insert*, *update* i *delete*
set auto[commit] off	(*default*) nie ma automatycznego *commit*, *commit* wykonywany automatyczne po naciśnięciu *Cntrl-Z* w trakcie wykonywania *drop*, *alter* i *create* oraz przy normalnym zakończeniu pracy z SQLPLUS (*exit)*

Spójność danych

- wszyscy użytkownicy *czytający dane* widzą dane takie, jakie były w momencie ostaniego zatwierdzenia
- tylko użytkownik *modyfikujący dane* widzi dane zmodyfikowane - zmodyfikowane dane będą dostępne dla innych użytkowników w momecie ich zatwierdzenia

możemy sobie zapewnić spójność *wielu* baz danych rozkazem

SET TRANSACTION READ ONLY

używany wtedy, gdy chcemy tworzyć raporty bazujące na więcej niż jedym zapytaniu do jednej lub wielu baz a jednocześnie chcemy zapewnić innym użytkownikom możliwośc modyfikacji tych baz.

- musi być pierwszym rozkazem transakcji
- w transakcji *READ ONLY* można używać tylko rozkazów *select* - nie można używać DML, *select for update* powoduje błąd
- rozkazy *commit*, *rollback* lub dowolny rozkaz DDL zamykają transakcję *READ ONLY*. Jeżeli był to rozkaz DDL użytkownik nie jest informowany o zamknięciu transakcji
- podczas działania transakcji *READ ONLY* wszystkie *select* odnoszą się do '*fotografi*' danych z początku transakcji
- inni użytkownicy nie są ograniczani - mogą czytać lub modyfikować dane w tabelach

Blokowanie tablic i rekordów

- każda blokada jest zwalniana w momencie zakończenia transakcji - zarówno przez *commit* jak i przez *rollback*

- rodzaje blokowania:
  - blokowanie DDL:
    - zabezpiecza dostęp do definicji obiektów
    - pozwala wykonywać modyfikacje słownika danych (*create table*, *alter table*, *drop table*)
    - wykonywane automatycznie przez system zarządzania bazą danych
  - blokowanie DML:
    - zabezpiecza dostęp do tabel użytkownika
    - kiedy trzeba - realizowane niejawnie przez system zarządzania bazą danych
    - użytkownik może *jawnie* zablokować tabele lub ich części
    - poziomy blokowania:
      - tabel - nakładane na całe tabele
      - wierszy - nakładane na wybrane wiersze w tabeli

jeżeli *Oracle* został zainstalowany z opcją TPO (*Transaction Processing Option)* to standardowo (automatycznie) realizowane są (z punktu widzenia użytkownika) następujące blokowania:

Rozkaz SQL	**poziom blokowania**	
	wiersz	**tabela**
SELECT	-	-
INSERT	X	RX
UPDATE	X	RX
DELETE	X	RX
DDL	-	X

-	brak blokowania
X	dopuszcza wykonywanie zapytań do zablokowanego obiektu lecz nie zezwala na wykonywanie innych działań na obiekcie
RX	umożliwia jednoczesny dostęp do tabeli wielu użytkownikom na następujących zasadach: • nie pozwala na nałożenie bloakdy na całą tabelę przez innych użytkowników • pozostali użytkownicy mogą blokować poszczególne wiersze tabeli i zatwierdzać zmiany • zablokowane wiersze można oglądać (*select*) lecz ich zmiana jest możliwa dopiero po zdjęciu blokady

Jawne blokowanie wierszy

- **select for update** - pozwala wybrać i zablokować wiesze które potem będziemy modyfikować. Blokowanie jest zwalniane po zatwierdzeniu lub wycofaniu zmian.

select ...
...
for update of [nowait]

'***nowait***' powoduje że rozkaz zostanie zakończony z błędem jeżeli nie można odrazu zablokować żądanych wierszy (są blokowane przez innego użytkownika). Jeżeli go nie użyjemy - Oracle będzie czekał na zdjęcie blokady. **Okres oczekiwania jest nieograniczony!**

ograniczenie *select for update*:

- nie wolno w nim użyć ***distinct*** i ***group by***
- nie wolno w nim użyć operatorów działań na zbiorach: ***union***, ***intersect***, ***minus***
- nie wolno w nim użyć funkcji grupowych (***count***, ***max***, ***avg***, ...)

Jawna blokada całej tablicy

rozkaz

LOCK TABLE *nazwa* IN *tryb* MODE [NOWAIT];

powoduje zdefiniowanie standardowej blokady nakładanej przez Oracle na tabele. Możliwe tryby:

row share	nie można blokować całej tabeli
row exclusive	tylko jeden użytkownik ma w danym momencie dostęp do wiersza
share update	nie można blokować całej tabeli
share	pozwala na jednoczesny dostęp ale bez wykonywania *update*
share row exclusive	to samo co ***share***
exclusive	tylko jdene użytkownik może blokować - blokowanie może dotyczyć tylko całej tabeli

nowait działa jak przy *select ... for update*.

Rozkaz SQL	poziom blokowania	
	wiersz	tabela
SELECT	-	-
SELCT FOR UPDATE	X	RS
LOCK tabela IN ... EXCLUSIVE	-	X
SHARE UPDATE	-	RS
ROW SHARE	-	RS
SHARE ROW EXCLUSIVE	-	SRX
ROW EXCLUSIVE	-	RX
SHARE	-	S
INSERT	X	RX
UPDATE	X	RX
DELETE	X	RX
DDL	-	X

-	brak blokowania
X	obiekt zarezerwowany tylko przez jednego użytkownika - tylko jeden użytkownik może zmieniać obiekt, inni mogą go tylko czytać
S	• zapobiega modyfikowaniu tabeli • dane mogą być odczytywane przez innych użytkowników
RS	• uniemożliwia blokowanie tabeli przez innych użytkowników w trybie *wyłączny* • wielu użytkowników może jednocześnie blokować wybrane wiersze • wszyscy mogą czytać dane
RX	• uniemożliwia blokowanie tabeli przez innych użytkowników w trybie *wyłączny* • wielu użytkowników może jednocześnie blokować wybrane wiersze • wszyscy mogą czytać dane • blokada typu *wspólna* nie jest możliwa
SRX	• uniemożliwia nałożenie blokady typu RS • tabela nie może być zablokowana w trybie *wyłączny* • tylko jeden użytkownik może zatwierdzać zmiany w tabeli, pozostali czekają w kolejce • w danym momencie tylko jeden użytkownik może usyskać tego typu blokadę (SRX) • blokada typu *wspólna* nie jest możliwa

ROWID

dla każdej tabeli istnieje unikalny numer wiersza - ***rowid*** - generowany przez *Oracle*. Tę kolumnę - ***rowid*** - można używać w podzapytaniu w celu wygenerowania numerów wierszy do których potem będziemy się starać odwołać (*select ... for update ...*). Wyszukiwanie wierszy przez ***rowid*** jest najszybszą metodą selekcji wierszy.

Tylko użycie *select ... for update ...* umożliwia posłużenie się ***rowid*** - w innym wypadku *Oracle* może zmienić jego wartość (***rowid*** jest fizycznym adresem wiersza w bazie - a adresy te mogą się zmieniać).

'deadlock' (zakleszczenia)

użytkownik X		**użytkownik Y**	
update	***emp***	***update***	***dept***
set	***sal = 1200,***	***set***	***loc = 'LONDON''***
where	***ename = 'LEWIS';***	***where***	***deptno = 20;***
update	***dept***	***update***	***emp***
set	***loc = 'RICHMOND'***	***set***	***sal = 1750***
where	***deptno = 20;***	***where***	***ename = 'LEWIS';***

spowoduje *deadlock*. *Oracle* rozpoznaje fakt wystąpienia *deadlocku* i sygnalizuje to jednemu z użytkowników wycofując (*rollback*) jednocześnie ostatni rozkaz jego transakcji. Użytkownik powinien w sposób jawny wycofać cała transakcję - może ją spróbować ponowić po pewnym czasie.

Perspektywy

Tworzenie:

```
create view     nazwa_perspektywy
                [ ( kolumna, ... ) ]
as select       ...
[with check option [constraint ograniczenie ] ]
```

- pozwala tworzyć 'pseudo-tablice', możliwe składanie elementów różnych tablic
- używana dokładnie tak samo jak zwykła tablica
- składowymi perspektywy mogą być kolumny i wyrażenia stworzone z kolumn tablic lub innych perspektyw
- kolumny tworzonej perspektywy mogą być kolumnami tablic lub wyrażeniami
- nie można używać ***order by*** przy tworzeniu perspektywy, można używać przy selekcji danych z perspektywy
- w stosunku do perspektywy można normalnie używać rozkazów DML - ***INSERT, UPDATE***
- jeżeli perspektywę tworzymy bez ***'with check option'*** - nie działa sprawdzanie poprawności wpisywanych danych. Powoduje to również wpisywanie błędnych danych do tabel, z których powstała perspektywa. O ile dopisane wartości nie będą spełniać warunków ograniczających perspektywę będą dołączone do tablic bazowych ale nie będą wyświetlane w perspektywie.

```
create view dept_summary
        (name, minsal, maxsal, avgsal )
as select
        ddname(emp.deptno ),
        min(sal), max(sal), avg(sal)
        from emp, dept
where emp.deptno = dept.deptno
group by emp.deptno;
```

```
create or replace function ddname( dp number )
        return varchar2
as
        begin
                declare dn varchar2(14);
        begin
                select dname into dn
                from dept
                where deptno = dp;
                return( dn);
        end;
        end;
```

- można ograniczyć czas dostępu do danych - za pomocą perspektywy:

```
create viewemp_details
as
select          empno, ename, job, deptno
from            emp
where           ename = user
    and     to_char( sysdate, 'HH24' ) between 9 and 17
    and     to_char(sysdate, 'D' ) between 2 and 6;
```

podczas tworzenia perspektywy (***create view***) ***select*** występujący po ***as*** nie jest wykonywany - jest tylko zapisany w słowniku danych. Prawa dostępu do tablic też nie są sprawdzane w momencie tworzenia perspektywy - są sprawdzane dopiero w momencie pobierania lub modyfikowania danych za pomocą perspektywy. Wynika stąd, że twórca perspektywy *nie musi mieć zagwarantowanych praw dostępu do tabel, z których tworzy perspektywę!!!* Jest to potencjalna '***dziura***' w systemie zabezpieczeń Oracla.

dane o perspektywach są pamiętane w tabeli USER_VIEWS.

Ograniczenia dotyczące modyfikowania danych za pomocą perspektyw:

- **DELETE nie jest dozwolone, jeżeli perspektywa zawiera:**
  - warunek łączący (tabele)
  - funkcje grupowe
  - klauzulę GROUP BY
  - kwalifikator ***distinct***
  - pseudokolumnę ***rownum*** (rownum zwraca numer (kolejność) w jakim wiersz był wybierany podczas selekcji danych z tabeli)
  - skorelowane zapytania

- **UPDATE nie jest dozwolone, jeżeli perspektywa zawiera:**
  - jakąkolwiek z opcji wymienionych dla DELETE
  - kolumnę tworzoną za pomocą wyrażenia

- **INSERT nie jest dozwolone, jeżeli perspektywa zawiera:**
  - jakąkolwiek z opcji wymioenionych dla UPDATE
  - kolumna z atrybutem NOT NULL, znajdująca się w tablicy z jakiej tworzona jest perspektywa nie jest składnikiem perspektywy

usuwanie perspektyw

komenda

drop view nazwa_perspektywy;

usuwa perspektywę. Perspektywę może usunąć tylko jej twórca. Usunięcie perspektywy może spowodować, że inne perspektywy, zbudowane na bazie usuwanej perspektywy staną się błędne

można użyć rozkazu

create or replace view ***nazwa***
as
select **...**

żeby usunąć i ponownie założyć perspektywę bez zmiany nadanych wcześniej praw dostępu do perspektywy.

prawa dostępu

GRANT - przydzielanie praw

- przez administratora systemu (użytkownik ***oracle*** lub ***dba***) - przydziela użytkownikowi zakres dostępnych czynności:

  grant *opcja* to u*żytkownik*;

możliwe opcje:

connect	możliwość dołączania się do Oracle ('login'), dostęp do obiektów na które ktoś pozwolił użytkownikowi, możliwość tworzenia synonimów i przekrojów
resource	możliwość tworzenia tabel, sekwencji i indeksów
dba	możliwość zakładania nowych użytkowników i możliwość dostępu do obiektów zastrzeżonych przez innych, zwykłych użytkowników

- przez twórcę tablicy lub perspektywy - aby zdefiniować prawa dostępu

grant *prawa* on *obiekt* to *użytkownik* [with grant option];

możliwe prawa:

select	możliwość selekcji danych z tabeli
insert	możliwość dopisania danych do tabeli
update	możliwość poprawienia danych w tabeli
delete	możliwość skreślenia danych z tabeli
alter	możliwość zmiany struktury tabeli (*alter table ...*)
index	tworzenie indeksów na tabeli
references	odwołanie do tabeli z 'ograniczeniami' (*constraints ?*)
all	wszystkie prawa

prawo	tabele	views	sequences
select	**x**	**x**	**x**
insert	**x**	**x**	
update	**x**	**x**	
delete	**x**	**x**	
alter	**x**		**x**
index	**x**		
references	**x**		

istnieje użytkownik ***public*** (== wszyscy). Można pozwolić na dostęp tylko do pewnych kolumn:

```
grant       update (dname, loc)
on          dept
to          adams, jones;
```

jednym rozkazem można przydzielić prawa kilku użytkownikom, można również przydzielić kilka rodzajów praw - nazwy użytkowników i rodzaje praw oddzielamy przecinkami.

komunikat '***table or view does not exist***' oznacza że nie istnieje obiekt, do którego się odwołujemy lub że nie mammy do niego praw dostępu dotyczących operacji, którą chcemy wykonać.

odbieranie praw

```
revoke      prawa
on          tabela lub przekrój
from        użytkownicy
```

można odebrać wszykie prawa ('***all***') wszystkim użytkownikom (***'public'***). Informacje o prawach dostępu do własnych tabel mam w perspektywach USER_TAB_PRIVS i USER_COL_PRIVS

synonimy

Dla każdej tabeli lub perspektywy do której mamy prawa dostępu możemy zdefiniować synonim:

```
create synonym          nazwa
for                     obiekt;

create public synonym   nazwa
for                     obiekt
```

pozwala to zastąpić konstrukcje

```
select      *
from        scott.emp;
```

konstrukcją

```
create synonym      semp
for                 scott.emp;

        ...

select              *
from                semp;
```

tylko administrator może zakładać i usuwać synonimy publiczne. Uswamy synonim poprzez

drop [public] synonym ***nazwa_synonimu;***

zakładanie nowego użytkownika (trzeba mieć przywilej **dba**)

create user ***nazwa*** **identified by** ***hasło*****;**

po założeniu użytkownika trzeba mu nadać prawa - zawsze ***connect***, zwykle również ***resource***. Każdy użytkownik może zmienić swoje hasło - komendą

alter user ***nazwa*** **identified by** ***hasło*****;**

wybrane tablice, perspektywy i synonimy 'systemowe':

Nazwa:	***Znaczenie:***
CAT	synonium dla USER_CATALOG
COLS	synonim dla USER_TAB_COLUMNS.
DBA_CATALOG	wszystkie tabele, widoki, synonimy i sekwencje
DBA_COL_COMMENTS	komentarze dla wszystkich tabel i widoków
DBA_COL_PRIVS	wszystkie prawa dostępu do kolumn
DBA_CONSTRAINTS	ograniczenia ('*constraints*') zdefiniowane dla bieżąco dostępnych tabel
DBA_INDEXES	opisy wszystkich indeksów
DBA_LOCKS	wszystkie bieżąco aktywne i zażądane blokady
DBA_OBJECTS	wszystkie obiekty w bazie danych
DBA_OBJECT_SIZE	wszystkie obiekty PL/SQL
DBA_ROLES	wszystkie bieżąco zdefiniowane role
DBA_SEQUENCES	wszystkie sekwencje zdefinowane w bazie
DBA_SNAPSHOTS	wszystkie 'snapshoty' w bazie
DBA_SOURCE	teksty źródłowy wszystkich obiektów pamiętanych w bazie
DBA_SYNONYMS	wszystkie synonimy
DBA_TABLES	opis wszystkich tabel w bazie danych
DBA_TAB_COLUMNS	wszystkie kolumny bieżąco dostępnych tabel i widoków
DBA_TAB_COMMENTS	komentarze do tabel i widoków
DBA_TRIGGERS	opisy wszystkich triggerów w bazie danych
DBA_USERS	informacja o wszystkich użytkownikach
DBA_VIEWS	teksty wszystkich zdefiniowanych widoków
DICT	Synonim dla DICTIONARY.
DICTIONARY	opisy struktur tabel i widoków
IND	Synonim dla USER_INDEXES.
OBJ	Synonim dla USER_OBJECTS.
SEQ	Synonim dla USER_SEQUENCES.
SESSION_PRIVS	przywileje i prawa dostępu bieżąco dostępne dla użytkownika
SYN	Synonim dla USER_SYNONYMS.
TABS	Synonim dla USER_TABLES.
V$NLS_PARAMETERS	bieżące wartości parametrów NLS ('wersje narodowe')
V$PARAMETER	bieżące wartości parametrów
V$SESSION	informacje o wszystkich bieżąco aktywnych sesjach
V$TIMER	bieżący czas, w setnych sekundy

generator sekwencji

istnieje możliwość generowania unikalnych numerów ('*sekwencji*')

create sequence ***nazwa***
[increment by *n*]
[start with *n*]
[maxvalue *n*] [nomaxvalue]
[minvalue *n*] [nominvalue]

nazwa	nazwa sekwecji - jak każda nazwa w Oracle
incremented by *n*	przyrost wartości. Może być dodatni lub ujemny, defalut: 1
start with *n*	wartość początkowa, default: 1 dla sekwencji rosnących, MAXVALUE dla sekwencji malejących
maxvalue *n* **nomaxvalue**	maksymalna generowana wartość, default: 1 dla sekwencji malejących, 10^{27}-1 dla rosnących
minvalue *n* **nominvalue**	minimalna wartość generowana, default: 1 dla sekwencji rosnących, 10^{27}-1 dla malejących

aby móc tworzyć sekwencje trzeba mieć przywilej ***resource***.

zasady używania sekwencji:

- najpierw musimy stworzyć sekwencję (***create sequence***)
- do sekwencji odwołujemy się poprzez:

nazwa_sekw.nextval	powoduje wygenerowanie nowej (kolejnej) wartości sekwencji i zwrócenie jej jako wartości wyrażenia *nazwa_sekw.nextval*; użycie kilku odwołań *nazw-sekw.nextvall* w obrębie tej samej instrukcji DML nie powoduje krotnej zmiany wartości sekwencji. Konstrukcja *nazwa_sekw.nextval* jest konstrukcją typu *++val*
nazwa_sekw.currval	zwraca wartość bieżącą sekwencji o podanej nazwie

- Oracle zapewnia unikalność generowanych wartości sekwencji również podczas współużywania jednej sekwencji przez grupę użytkowników - może to spowodować że użytkownik nie otrzyma *kolejnych* numerów generowanych przez sekwencję - ale na pewno będą one unikalne
- ***rollback*** nie 'cofa' wartości sekwencji
- 'pad komputera' może spowodować powstanie 'dziur' w wartościach generowanych przez sekwencję

- pseudokolumny *nazwa_sekw.nextval* i *nazwa_sekw.currval* mogą być stosowane w:
  - klauzuli ***select*** rozkazu **select** (z wyjątkiem perspektyw)
  - liście wartości rozkazu INSERT
  - klauzuli SET rozkazu UPDATE
  - najbardziej zewnętrzym zapytaniu w złożonym rozkazie ***select***

- pseudokolumny *nazwa_sekw.nextval* i *nazwa_sekw.currval* nie mogą być stosowane w:
  - klauzuli ***select*** definiującej przekró (***create view***)
  - z kwalifikatorem ***distinct***
  - jeżeli występują klauzule ***order by, group by, connect by*** lub ***having***
  - z operatorami ***union, intersection*** lub ***minus***
  - w podzapytaniu

Właściciel sekwencji może nadać lub odebrać prawa używania sekwencji innym użytkownikom. Dla sekwencji można nadawać prawa ***alter*** i ***select***. Można nadać prawa do selekcji z klauzulą ***with grant option***.

modyfikacja sekwencji

alter sequence *nazwa*
[increment by *n*]
[maxvalue *n*] [nomaxvalue]
[minvalue *n*] [nominvalue]

- nowa definicja sekwencji obowiązuje dla numerów generowanych po wykonaniu zmiany definicji sekwencji
- badana jest poprawnośc zmian definicji sekwencji (np. nowe *maxvalue* nie może być mniejsze od ostatnio wygenerowanej wartości)
- ***alter sequence*** nie może zmienić parametru ***start with***
- żeby zacząć generować sekwencję od nowej wartości początkowej trzeba skreślić i ponownie założyć sekwencję

usuwanie sekwencji

drop sequence *nazwa:*

Definicje sekwencji są pamiętane w tabelach USER_SEQUENCES i ALL_SEQUENCES.

indeksy

indeksy są używane przez Oracle automatycznie - użytkownik może nawet nie wiedzieć, że ***select*** którego używa odwołuje się do indeksów. Natomiast tworzenie indeksów powinno zostać zainicjowane rozkazem

create [unique] index ***nazwa***
on ***tablica(kolumna, ...)*;**

możliwe rodzaje indeksów:

unique	jeżeli wystąpiła klauzula ***unique*** - Oracle dopuszcza tylko unikalne wartości w kolumnach, tworzących indeks
non unique	default, wartości w kolumnach tworzących indek nie muszą być unikalne. Zapewnia najszybszy dostęp do danych
single column	indeks tworzony tylko na jednej kolumnie
concatenated	indeksowanie do 16 kolumn, może poprawić prędkość przeszukiwania lub zapewnić unikalność kombinacji wartości kolumn

te dwie klasy są niezależne - można je łączyć (***unique single column***, ***unique concatenated***, ...)

- można mieszać typy danych podczas tworzenia indeksu skonkatenowanego
- w indeksach nie można używać wyrażeń - tylko nazw kolumn
- indeks może być tworzony *on-line* - podczas pracy z bazą
- jeżeli utworzymy indeks ***unique concatenated*** to zapewni nam to niepowtarzalność danych w obrębie skonkatenowanych kolumn

create unique index ***order***
on ***shipments(s_num, p_num);***

usuwanie indeksów

drop index ***nazwa*;**

indeks zostanie użyty przez Oracle automatycznie jeżeli:

- indeksowana kolumna wystąpi w klauzuli ***where***. Jeżeli w rozkazie ***select*** nie występuje klauzula ***where*** - indeks nie będzie używany
- indeksowana kolumna z klauzuli ***where*** nie występuje jako część wyrażenia - jest 'samodzielną' kolumną. Nawet konstrukcja

```
select      *
from        emp
where       upper( ename ) = 'JONES';
```

gdy '***ename***' jest kolumną według której poindeksowano tablicę EMP nie spowoduje użycia indeksu

zalecenia:

- opłaca się założyć indeks jeżeli tablica posiada więcej niż 200 wierszy
- zaleca się indeksowanie po kolumnach w których nie wsytępują często powtarzające się wartości
- zaleca się indeksowanie talic po kolumnach często używanych w klauzulach ***where*** oraz w warunkach łączących dwie tabele
- jeżeli dwie lub więcej kolumny często występują w klauzuli ***where*** - założyć indeks skonkatenowany
- unikać wiecej niż trzech indeksów zdefiniowanych dla tabeli - może to spowodować '*przeciążenie*' systemu. Nie dotyczy to przypadku, kiedy najczęsciej - lub jedynie - wykonujemy rozkaz ***select***
- w instrukcjach ***select*** działających na kilku tablicach - indeksy są używane tylko dla pierwszej tablicy - następne są sekwencyjnie przeszukiwane. Dlatego kolejnośc tabel w instrukcji ***select*** może znacząco wpłynąć na czas jej wykonania

instrukcje PL/SQL

program w PL/SQL składa się z bloków - dowolnie zagnieżdżonych. Każdy blok ma taką samą strukturę:

```
declare
    definicja zmiennych używanych w bloku
begin
    instrukcje
exception
    obsługa przerwań, wygenerowanych podczas wykonywania programu
end;
```

- bloki mogą być ***anonimowe*** (bez nazwy) - są traktowane jak instrukcje złożone - lub ***nazwane*** - są wtedy traktowane jak procedury lub funkcje

```
declare
        qty_on_hand   number(5);
begin
        select        quantity into qty_on_hand
        from          inventory
        where         product = 'TENNIS RACKET';

        if qty_on_hand > 0 then
                update        inventory
                              set quantity = quantity - 1
                where         product = 'TENNIS RACKET';

                insert        into purchase_record
                values        ( 'Tennis Racket purchased', sysdate );
        else
                insert        into purchase_record
                values        ('Out of Tennis Rackets', sysdate );
        endif;

        commit;

exception
        when   no_data_found  then
                insert   into error_table
                values  ('Product TENNIS RACKET not found' );
end;
```

zasady ogólne:

- obowiązują wszystkie zasady tworzenia instrukcji SQLa
- słowa zarezerwowane mogą być używane jako identyfikatory tylko wtedy kiedy są ujęte w cudzysłowy - nie zalecane
- komentarze:

```
/*
     ... wieloliniowe ...
*/

instrukcja    -- do końca lini
```

- najbardziej zewnętrzny blok nie może być etykietowany
- bloki można dowolnie zagnieżdżać w sobie. Sekcja ***exception*** też może zawierać zagnieżdżone bloki
- **instrukcje z bloków PL/SQL nie mają bezpośredniego dostępu do ekranu!!!** (wypisywanie wartości - poprzez wpisywanie do tabeli). Instrukcje PL/SQL używane wenątrz formatek tworzonych za pomocą *Oracle Forms* mogą wyświetlać informacje wprost na ekranie.
- w blokach PL/SQL nie można używać instrukcji ***accept***
- wewnątrz bloków PL/SQl nie mogą wystąpić instrukcje DDL
- instrukcja ***select***, występująca wewnątrz bloku PL/SQL spowoduje ***exception*** jeżeli wyselekcjonuje więcej niż jeden wiersz
- instrukcje DML modyfikujące dane mogą przetwarzać wiele wierszy
- nie powinno się używać zmiennych o takich samych nazwach jak nazwy kolumn tablicy bo Oracle będzie taką nazwę traktował jako nazwę kolumny
- blokowanie danych, spowodowane używaniem instrukcji DML działa w bloku PL/SQL standardowo - blokady są zwalniane na końcu transakcji
- sterowanie transakcjami w bloku PL/SQL działa tak samo jak w SQL
- włączenie trybu ***autocommit*** spowoduje potraktowanie całego bloku PL/SQL jako pojedyńczej instrukcji

deklaracja zmiennych

nazwa [constant] typ [not null] [:= wartośc_początkowa]

- klauzula ***constant****i definiuje stałą*
- istnieje dodatkowy typ - ***boolean*** - którego można używać w blokach PL/SQLa. Dopuszczalne wartości to TRUE, FALSE i wynik wyrażenia logicznego
- jako ***typ*** może wystąpić również konstrukcja

***nazwa_tabeli.nazwa_kolumny*%TYPE**

powodująca utworzenie zmiennej o tykie zgodnym z typem kolumny w tabeli. Podstawianie typu natępuje wtedy podczas kompilacji bloku PL/SQL - mogą wystąpić błędy wykonania jeżeli pomiędzy kompilacją a wykonaniem bloku zmodyfikujemy typ kolumny w tabeli (***alter table***)

- użycie klauzuli ***not null*** powoduje konieczność podstawienia wartości początkowej
- przekroczenie rozmiaru deklarowanej zmiennej (***char(n), number(n)***) powoduje ***exception***
- obowiązują zwykłe zasady 'przykrywania' zmiennych i zakresy 'działania' zmiennych
- do zmiennych z bloku 'wyższego poziomu' można się odwoływać poprzedzając nazwę zmiennej zmienną bloku z wyższego poziomu. Nazwa bloku występuje jako etykieta

rekordy

składnia

declare
nazwa_zmiennej ***nazwa_tabeli*****%ROWTYPE;**

deklaruje zmienną będącą rekordem o polach takich, jak kolumny w tabeli o definicję której jest on oparty. Rekordy można definiować na bazie tabel, pespektyw lub kursorów (zob. dalej). Odwolanie - poprzez 'notację kropkową':

nazwa_zmiennej.nazwa_pola

jeżeli tylko typy zmiennych rekordowych są zgodne - można podstawiać całe rekordy, bez wymieniania pól

declare
zm1 ***emp*****%ROWTYPE;**
zm2 ***emp*****%ROWTYPE;**

...

zm1 **:=** ***zm2*****;**

nie można użyć nazwy całego rekordu w lisćie ***values*** instrukcji ***insert*** - trzeba jawnie wymieniać poszczególne pola rekordu

podstawienia

postać:

zmienna := wyrażenie;

Oracle próbuje dokonać automatycznej konwersji typów jeżeli jest to konieczne (nawet podstawienie zapisanej znakowo - jako łstała znakowa - liczby pod zmienną numeryczną jest poprawne). Jeżeli wystąpi błąd konwersji - generowane jest przerwanie ***value_error***.

normalna kolejność wykonywania operatorów:

najpierw	**, NOT	potęgowanie
	+,-	identyczność, negacja
	*,/	
	+,-,||	dodawanie, odejmowanie, konkatenacja
	=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	porównywania
	AND	
na końcu	OR	

nawiasy (podwyrażenia) działają standardowo. Można używać standardowych funkcji SQLa - funkcji grupowych (***avg, min, max, count, sum, stddev*** i ***variance***) tylko w instukcjach ***select*** występujących wewnątrz bloku PL/SQL.

tworzenie bloku PL/SQL - zewnętrznym edytorem lub edytorem ***sqlplus*** - o ile zaczniemy wpisywanie od słowa ***declare*** i zakończymy linią zawierającą tylko kropkę w pierwszej kolumnie. Wykonanie - przez ***run*** lub /.

'niejawne kursory' - testowanie wyników komend SQL w blokach PL/SQL

wykonanie dowolnej komendy SQL w obrębie bloku PL/SQL powoduje nadanie wartości - zgodnie z wynikiem działania komendy SQL - następującym '*niejawnym kursorom*':

SQL%ROWCOUNT	liczba wierszy przetworzonych komendą SQL
SQL%FOUND	TRUE jeżeli przetworzono choć jeden wiersz, w przeciwnym przypadku: FALSE
SQL%NOTFOUND	***not*** **SQL%FOUND**

każda kolejna instrukcja SQL zmienia wartości tych zmiennych. '*Kursory niejawne*' mogą być testowane również w części ***exception*** bloku. Nie można wprost wstawić wartości *niejawnego kursora* do elementu tabeli - trzeba go podstawić do zmiennej roboczej a następnie zmienną roboczą wstawić do tabeli.

instrukcja *select* w blokach PL/SQL

postać:

```
select      element, ...
into        zmienna, ...
from        tabela, ...
[ where     warunki ... ]
[ group by  element, ... ]
[ having    warunki ... ]
[ for update ]
```

- klauzula ***into*** musi wystąpić - ilość zmiennych w ***into*** i ilość zmiennych selekcjonowanych przez ***select*** musi być ta sama
- jeżeli select nie wybieże dokładnie jednego elementu zostanie wygenerowane przerwanie:

no_data_found	nie znaleziono żadnego wiersza spełniającego warunki z ***select***
too_many_rows	wyselekcjonowano więcej niż jeden wiersz

obsługa wyjątków (*exceptions*)

wystąpienie błędu podczas wykonywania bloku PL/SQL powoduje wystąpienie wyjątku - jeżeli nie zdefiniujemy instukcji obsługujących ten wyjątek wykonywanie bloku PL/SQL zakończy się z błedem wykonania[5]. Postać instrukcji obsługi wyjątków:

```
exception
        when nazwa_wyjątku [ or nazwa_wyjątku ... ] then
                instrukcje ...
        when nazwa_wyjątku [ or nazwa_wyjątku ... ] then
                instrukcje ...

    ...

        [when others then
                instrukcje ...
        ]
```

niektóre wyjątki 'predefiniowane':

DUP_VAL_ON_INDEX	nie unikalna wartość w kolumnie indeksowej która ma atrybut ***unique***
INVALID_CURSOR	nielegalna wartość kursora (zob. dalej)
INVALID_NUMBER	nielegalna liczna (np. przy automatycznej konwersji)
LOGIN_DENIED	brak pozwolenia na ***login***
NO_DATA_FOUND	***select*** nie znalazł żanego wiersza z danymi
NOT_LOGGED_ON	użytkownik jeszcze nie zaloginowany
PROGRAM_ERROR	błąd wykonania programu
STORAGE_ERROR	błąd - zwykle przepełnienie - obszaru na dysku
TIMEOUT_ON_RESOURCE	zbyt długie oczekiwanie na zasób
TOO_MANY_ROWS	***select*** wybrał więcej niż jeden wiersz
VALUE_ERROR	zła wartość (np. przy automatycznej konwersji lub podstawianiu daty)
ZERO_DIVIDE	próba dzielenia przez 0

dostępne są dwie fukcje systemowe zwracające informacje o zaistniałym błędzie:

SQLCODE	numer błędu - poza blokiem ***exception*** ma wartość zero
SQLERRM **SQLERRM(*n*)**	tekst komunikatu o błędzie który ostatnio wystąpił w tej postaci zwraca tekst komunikatu dla błędu o numerze *n*

komunikaty o błędach są pamiętane na plikach ****.msb***.

jeżeli w bloku na poziomie (zagnieżdżenia) na którym wystąpił wyjątek nie ma jego obsługi - wyjątek jest 'przekazywany' na poziom wyższy - aż do znalezioenia procedury obsługi wyjątku lub do przewania całego programu - jeżeli wyjątek nie jest nigdzie obsługiwany.

można zignorować ('zaślepić') wyjątek wpisując jako instrukcję ktora ma się wykonać po jego wystąpieniu **null**

[5]ale tylko na poziomie, na którym był wykonywany blok powodujący wystąpienie wyjątku. 'Wyższe' poziomy bloków PL/SQL wykonują się dalej.

można generować '*własne*' wyjątki:

deklaracja:

```
declare
        nazwa  exception;
```

wygenerowanie wyjątku:

```
raise nazwa_wyjątku;
```

obsługa - jak dla wyjątków 'systemowych'. Można również zdefiniować wartość wstawianą jako kod błedu do zmiennej **SQLCODE** po wystąpieniu wyjątku 'użytkownika':

```
declare
    fetch_failed    exception;
    pragma exception_init ( fetch_failed, -1002 );
```

kolejność istotna!!!

```
declare
    e_mess          char(80);
begin
    declare
        v1          number(4);
    begin
        select  empno into v1
        from    emp
        where   job = 'PRESIDENT';

    exception
        when    too_many_rows then
                insert  into job_errors
                values  ('More than one President'):
    end;

    declare
        v1          number(4);
    begin
        select  empno into v1
        from    emp
        where   job = 'MANAGER';

    exception
        when    too_many_rows then
                insert  into job_errors
                values  ('More than one Manager'):
    end;

exception
    when others then
        e_mess := substr( SQLERRM, 1, 80 );
        insert  into general_errors
        values  ( e_mess );
end.
```

instrukcja *if*

postać:

```
if   warunek        then
                instrukcje ...
[ elsif       warunek        then
                        instrukcje ...
  ...
]
[ else
                instrukcje ...
]
end if;
```

instrukcja pętli *loop*

postać:

```
[ << etykieta >> ]    loop
                                instrukcje ...
                                end loop;
```

pętla 'bezwarunkowa' ('*forever*'). Do zakończenia takiej pętli można użyć instrukcji

```
exit [ etykieta_pętli ] [ when warunek ];
```

- jeżeli podamy etykietę - zakończona zostanie pętla o tej etykiecie (zagnieżdżenia). Defaultowo - bieżąca pętla.
- jeżeli nie podamy ***when warunek*** - pętla zostanie zakończona bezwarunkowow

pętla *for*

postać:

```
for zmienna_sterująca in [ reverse ] wartość_początkowa .. wartość_końcowa
    loop
            instrukcje ...
    end loop;
```

- zawsze wykonywana z krokiem '1'
- ***reverse*** powoduje zminiejszanie wartości zmiennej kontrolnej pętli od ***wartościi_początkowej*** do ***wartościi_końcowej***
- ***zmienna_sterująca*** jest zmienną roboczą automatycznie tworzoną na czas działania pętli

pętla *while*

postać:

```
while       warunek
    loop
            instrukcje ...
    end loop;
```

- zakończenie pętli ***while*** - kiedy ***warunek*** będzie miał wartość FALSE
- jeżeli przed wykonaniem pętli ***warunek*** będzie miał wartość FALSE - pętla nie będzie ani raz wykonana

instrukcja *go to*

postać:

```
goto etykieta;
```

```
begin
<<block1>>    declare
                        var1    number;

                    begin
                        <<block2>>    declare
                                            var1 number := 400;
                                      begin
                                            block1.var1 := block1.var1 + 1;
                                      end block2;  -- traktowana jak komentarz
                    end block1;
<<main>>      loop
                        ...
                        loop
                            ...
                            exit main when total_done = 'YES';
                            exit when inner_done = 'YES';
                        end loop;
                    end loop main;
end;
```

kursory '*jawne*'

używane w blokach PL/SQL do przyspieszenia wyszukiwania rekordów. Nie generują żadnych ***exception*** - nawet kiedy nic nie zostanie wyselekcjonowane - poza ***fetch_out_of_sequence*** generowanego kiedy próbujemy wykonać ***fetch*** gdy już nie ma więcej wyselekcjonowanych kursorem danych.

deklaracja kursora:

cursor ***nazwa*** **[(** ***parametry*****, ...)]** **is** ***wyrażenie_typu_select*****;**

zapamiętuje definicję kursora. '***wyrażeniem_typu_select***' może być zwykłą instrukcją ***select*** nie posiadającą klauzuli ***into***. Ewentualne parametry służą do 'dodefiniowania' instrukcji select podczas otwierania kursora

otwarcie kursora:

open ***nazwa_kursora*** **[(** ***lista argumentów*** **)];**

w momenice wykonywania ***open*** sprawdza się, jakie wiersze spełniają warunek definiujący kursor - tylko te wiersze, z wartościami z momentu ***open*** są 'dla mnie' pamiętane i udostępniane poprzez kursor.

po otwarciu kursora kursor wskazuje pierwszy wybrany wiersz.

pobieranie wartości:

fetch ***nazwa_kursora*** **into** ***zmienna*****, ... ;**

powoduje pobranie danych z kolejnego wiersza wyselekcjonowanego przez kursor i wstawienie tych danych do zmiennych. Ilość zmiennych musi być zgodna z ilością kolumn z definicji kursora. Jeżeli ***fetch*** nie wybierze żadnej (pierwszej lub kolejnej) wartości - zmienne otrzymają wartości ***null***.

zamknięcie kursora:

close ***nazwa_kursora*****;**

zamyka wskazany kursor - umożliwia jego ponowne otwarcie.

zmienne 'systemowe' związane z kursorem:

kursor**%FOUND**	TRUE jeżeli ostatnia instrukcja ***fetch*** spowodowała wczytanie nowych danych, w przeciwnym razie FALSE
kursor**%NOTFOUND**	odwrotność ***kursor*****%FOUND**
kursor**%ROWCOUNT**	zwraca ilość wierszy wybranych do tego momentu z aktywnego kursora
kursor**%ISOPEN**	TRUE jeżeli kursor jest otwarty, w przeciwnym razie FALSE

- nie można się 'cofnąć' po kursorze!!!

- można odwołać się do ostatnio wyselekcjonowanego kursorem rekordu używając klauzuli **where current of *nazwa_kursora*** w instrukcji DML

```
/*      przetwarzaj wiersze z tabeli DEPT przenosząc departamenty SALES do
        Dallas a pozostałe do Nowego Jorku. Zliczaj ilość przemieszczonych
        departamentów.
*/
declare
                cursor          c1 is
                                select  dname, loc
                                from    dept
                                for update of loc;
        dept_rec                c1%ROWTYPE;
        sales_count     number := 0;
        non_sales               number := 0;
begin
        open c1;
        loop
                fetch   c1 into dept_rec;
                exit    when c1%NOTFOUND;
                if      dept_rec.dname = 'SALES' and dept_rec.loc != 'DALLAS' then
                        update dept     set loc = 'DALAS'
                                where current of c1;
                        sales_sount := sales_count + 1;
                elsif   dept_rec.dname != 'SALES' and dept_rec.loc != 'NEW YORK' then
                        update dept     set loc = 'NEW YORK'
                                where current of c1;
                        non_sales := non_sales + 1;
                end if;
        end loop;
        close c1;
        insert  into counts( sales_set, non_sales_set )
                        values ( sales_count, non_sales );
        commit;
end;
```

istnieje specjalna odmiana pętli ***for*** służąca do poruszania się po kursorze:

```
for identyfikator_rekordu in identyfikator_kursora [ ( parametry, ... ) ]
    loop
        instrukcje ...
    end loop;
```

ta instrukcja:

- otwiera ('***open***') kursor o podanym identyfikatorze ('***identyfikator_kursora***')
- wprowadza (***fetch***) nowy wiersz przy każdej iteracji
- powoduje wyjście z pętli gdy wszystkie wiersze są przetworzone
- zamyka kursor

zmienna ***identyfikator_rekordu*** jest zmienną roboczą tworzoną automatycznie na czas działania pętli ***for*** - do niej wczytywane są wartości pobieranych rekordów

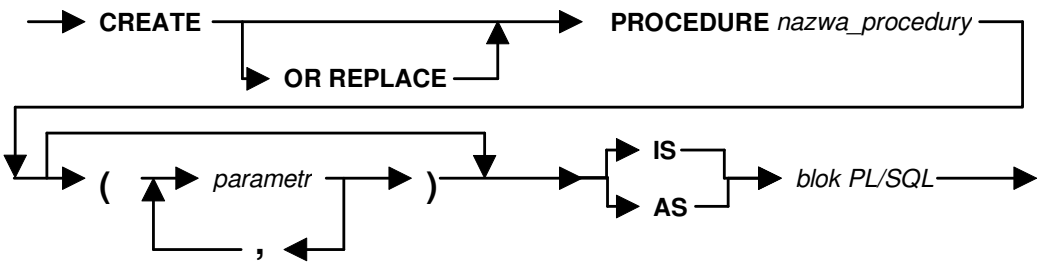
można tej konstrukcji użyć do 'niejawnego kursora' - bez jego definicji:

```
for nazwa_rekordu in ( wyrażenie_zapytania )
    loop
        instrukcje ...
    end loop;
```

'***wyrażenie_zapytania***' jest instrukcją ***select*** służącą do wyselekcjonowania danych

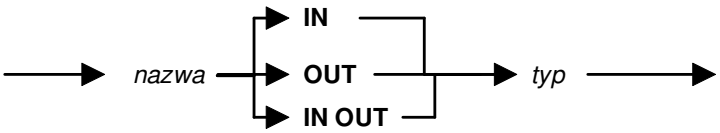
```
for rec in ( select ename from emp where empno = 10 )
    loop
        if rec.ename = 'JONES' then
            ...
    end loop;
```

Procedurê u¿ytkownika tworzymy wpisuj¹c do pliku z rozszerzeniem jej treœæ a nastêpnie "uruchamiając" (***start***, *@*) ten plik. Definicja procedury ma postaæ:



gdzie *"parametr*" oznacza:

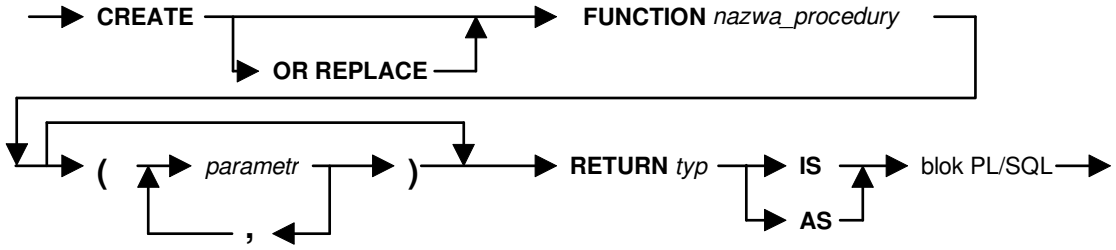
parametr



Parametr procedury lub funkcji może być jednego z typów:

IN *parametr wejściowy*, możemy tylko pobierać z niego wartość, nie może wystąpić po lewej stronie podstawienia
OUT *parametr wyjściowy*, możemy tylko wstawiać do niego wartość, nie może wystąpić po prawej stronie podstawienia
IN OUT *parametr wejściowo - wyjściowy*, możemy zarówno pobierać, jak i wstawiać do niego wartości

Definicja funkcji musi dodatkowo posiadać informację o typie zwracanej wartości:



Istnieje ograniczenie na postać instrukcji *select* używanej w procedurach i funkcjach PL/SQL. *Select* musi zwracać pojedynczą wartość - lub grupę wartości, podstawianą pod zmienną lub grupę zmiennych. Ma wtedy postać:

select ... *lista kolumn lub wyra¿eñ*... into ...*lista nazw zmiennych*...
from ...
where ...

Procedurę lub funkcję możemy przetestować używając instrukcji

execute *nazwa_procedury_lub_funkcji*

Zaleca się minimalizowanie ilości parametrów w procedurach i funkcjach poprzez używanie zmiennych systemowych (*sysdate, user*).

Przykład:

```
/*      =====  Funkcja get_sal zwraca zarobki pracownika którego numer jest jej
               parametrem. Jeżeli takiego pracownika nie ma - zwróć zero
*/
create or replace function get_sal( p_num in emp.empno%TYPE ) return number
as
begin
        declare
                p_sal    emp.sal%TYPE;
begin
        select sal into p_sal
        from emp
        where   empno = p_num;
        return( p_sal);

exception
        when no_data_found then
                return( 0 );
end;
end;
/
```

Jeżeli powyższe instrukcje wpiszemy do pliku *func01.sql* to uruchomienie tego pliku (***start func01*** lub ***@func01***) spowoduje skompilowanie funkcji *get_sal*. Oracle pamięta każdą procedurę i funkcję w dwóch postaciach: Źródłowej i półskompilowanej, kompilując automatycznie, jeżeli zmieni się jej kod źródłowy, a my odwołamy się do niej.

Aby przetestować działanie funkcji *get_sal* możemy użyć procedury ***put_line*** z pakietu bibliotecznego ***dbms_output***. Pakiet ten jest dostępny w procedurach i funkcjach wywoływanych z wnętrza ***sqlplus***. Procedur z tego pakietu nie możemy niestety używać wewnątrz bloków PL/SQL wewnątrz triggerów. Najcześciej używaną procedurą z tego pakietu jest ***put_line***, wywoływaną w następujący sposób:

execute dbms_output.put_line(*tekst*);

Pakiet jest aktywowany instrukcją

set serveroutput on

Aby przetestować funkcję *get_sal* możemy stworzyć plik (skrypt) zawierający następujące instrukcje:

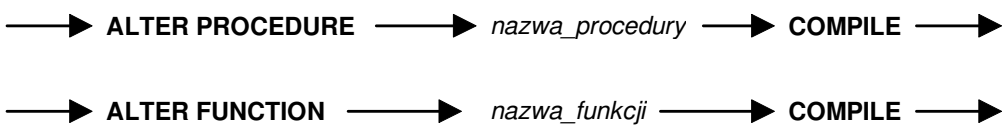
```
set echo off
set verify off
set serveroutput on
accept pn prompt "Podaj numer pracownika: "
execute dbms_output.put_line( 'Numer pracownika: ' || &pn );
execute dbms_output.put_line( 'Zarobki: ' || get_sal( &pn ) );
set serveroutput off
```

Zakładając, że ww. skrypt został zapisany na pliku *test01.sql* uruchomimy go instrukcją

start *test01*

Kod procedur i funkcji jest pamiętany w tabeli USER_SOURCE. Błędy kompilacji ostatnio kompilowanej procedury lub funkcji możemy wyświetlić komendą SHOW ERRORS. Komenda DESCRIBE *nazwa_procedury_lub_funkcji* wyświetla jej nagłówek.

Możemy *wymusić* skompilowanie procedury lub funkcji używając instrukcji:



Przekazywanie parametrów do procedur i funkcji

Parametry do procedur i funkcji możemy przekazywać na dwa sposoby:

- umieszczając odpowiednią ilość wyrażeń w wywołaniu procedury lub funkcji - musimy zachować zgodność ilości wyrażeń i parametrów formalnych oraz ich typów
- używając konstrukcji ***nazwa_parametru_formalnego => wyrażenie***. Możemy wtedy umieszczać wartości parametrów w dowolnej kolejności.

Można mieszać obie te metody.

Prawa dostępu do procedur i funkcji

Właściciel procedury lub funkcji (ten, kto ją stworzył) może ją udostępnić innym użytkownikom nadając im prawo dostępu ***execute***

grant execute on *nazwa_procedury_lub_funkcji* to *nazwa_użytkownika*

Użytkownik ***public*** ma standardowe znaczenie. Prawa można odebrać komendą ***revoke***.

Komunikacja z użytkownikiem w procedurach i funkcjach

Wewnątrz procedur i funkcji - szczególnie tych używanych w triggerach - nie możemy używać pakietu *dbms_output*. Komunikację z użytkownikiem możemy sobie zapewnić na jeden z dwóch sposobów:

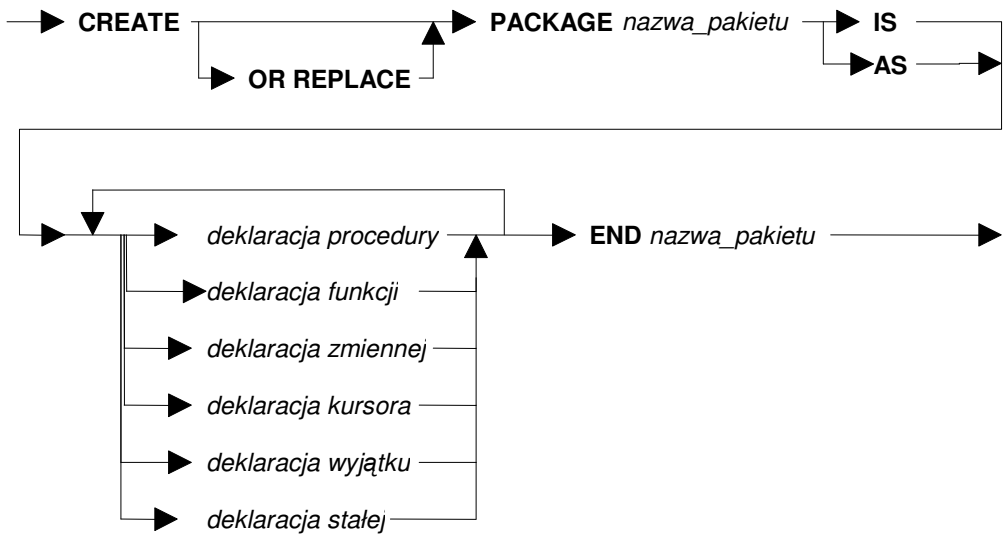
- stworzyć tabelę o dowolnej nazwie - np. *log* - zawierającą pojedynczą kolumnę tekstową, wystarczająco szeroką - by pomieściła tekst komunikatu. Wewnątrz instrukcji bloku PL/SQL wpisujemy do tej tabeli komunikaty do użytkownika, a w skrypcie uruchamiającym blok PL/SQL - lub wprost w ***sqlplus*** - wyświetlamy zawartość tej tabeli zwykłą instrukcją *select*.
- użyć procedury systemowej **raise_application_error**. Wywołujemy ją zawsze z dwoma parametrami:

raise_application_error(*numer_błędu*, *tekst_komunikatu_o_błędzie*)

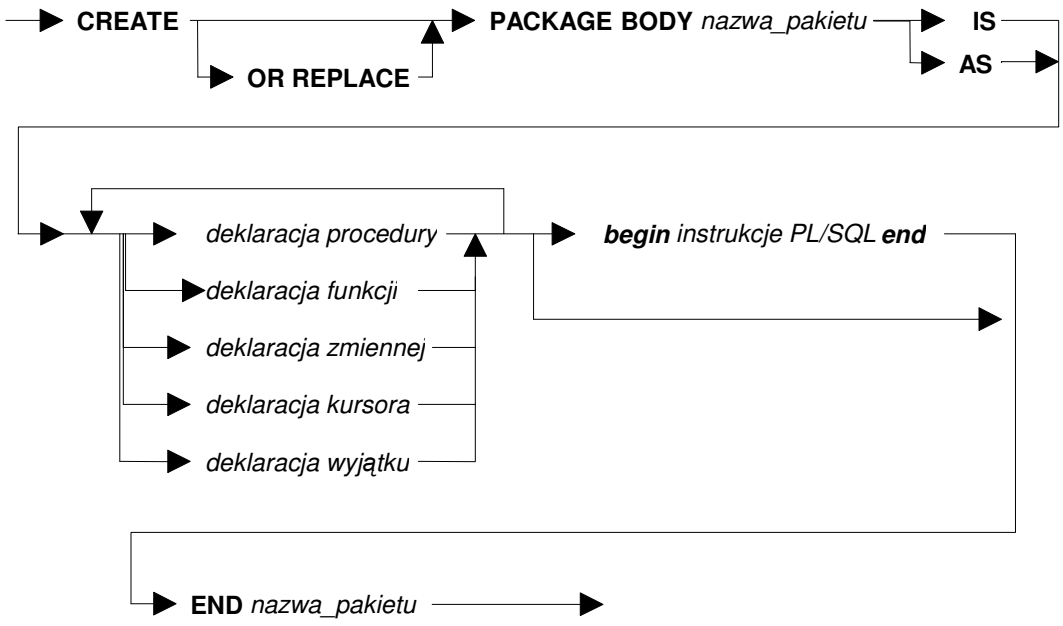
numer_błędu musi być liczbą z przedziału -20999 ÷ -20000 (ma wartość ujemną) - pozostałe numery są zarezerwowane dla błędów systemowych. *Tekst_komunikatu_o_błędzie* jest stałą tekstową. Użycie tej instrukcji spowoduje wygenerowanie błędu wykonania i wyświetlenie odpowiedniego komunikatu dla użytkownika. **raise_application_error** kończy wykonanie bloku PL/SQL.

Możemy stworzyć - z procedur i funkcji - ***pakiet***. Pakiet działa podobnie jak klasa w obiektowych językach programowania. Definicja pakietu składa się z dwóch części:

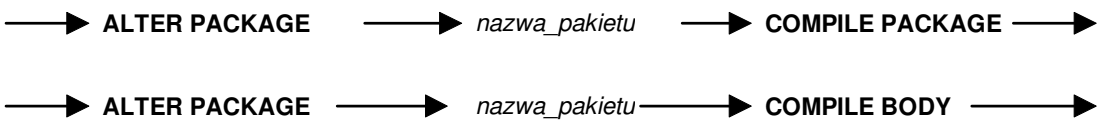
deklaracja obiektów dostępnych z zewnątrz:



deklaracja ciała pakietu oraz obiektów wewnętrznych, niedostępnych spoza pakietu:



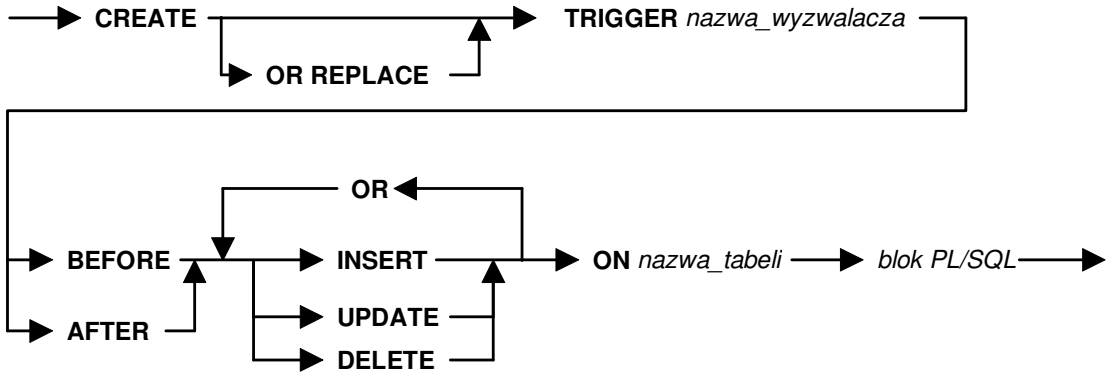
Pakiety tworzymy tak samo jak procedury i funkcje - wpisujemy je do pliku z rozszerzeniem *.sql* a następnie uruchamiamy taki plik (***start*** lub ***@***). Mo¿emy wymusić rekompilację definicji obiektów dostępnych z zewnątrz lub ciała pakietu instrukcjami



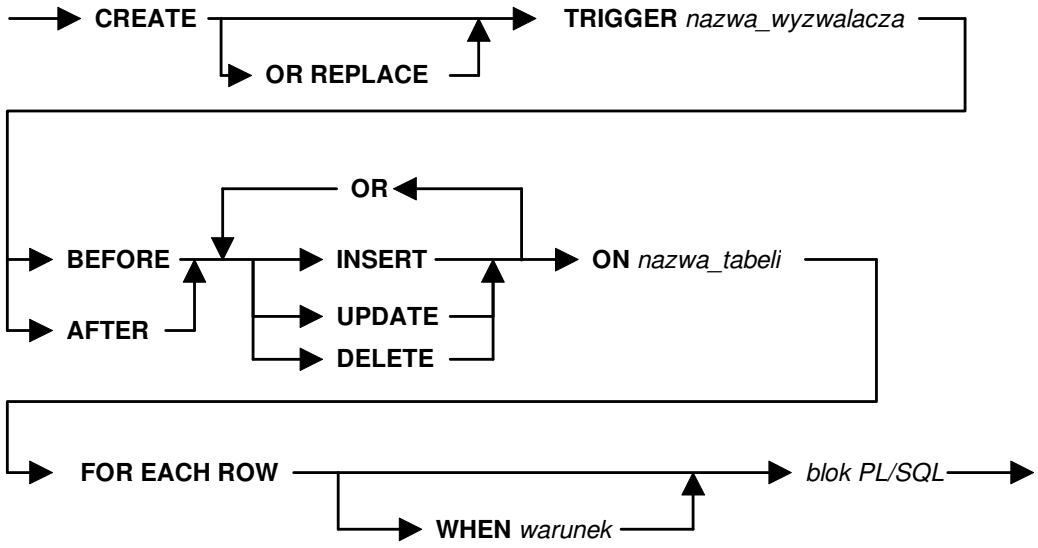
Procedury, funkcje i pakiety usuwamy - podobnie jak tabele - instrukcję ***drop***.

Triggery działają jak grupy instrukcji PL/SQL uruchamiane automatycznie po wystąpieniu pewnego zdarzenia. Na poziomie SQL dostępne są zawsze następujące triggery:

trigger 'jednorazowy' wykonywany tylko raz dla wszystkich instrukcji wchodzących w skład INSERT, UPDATE lub DELETE:



triggery wykonywane dla każdego rekordu przetwarzanego instrukcjami INSERT, UPDATE lub DELETE:



Wewnątrz bloków PL/SQL znajdujących się wewnątrz triggerów możemy - dodatkowo - używać następujących konstrukcji:

- w triggerach związanych z modyfikacją danych (***INSERT, UPDATE***) można odwoływać się zarówno do danych sprzed modyfikacji jak i danych zmodyfikowanych:

:old.*nazwa_kolumny*	umożliwia pobranie wartości kolumny *sprzed* modyfikacji
:new.*nazwa_kolumny*	umożliwia zmianę wartości zmodyfikowanej przez użytkownika lub wstawienie wartości początkowych do pól modyfikowanych przez użytkownika

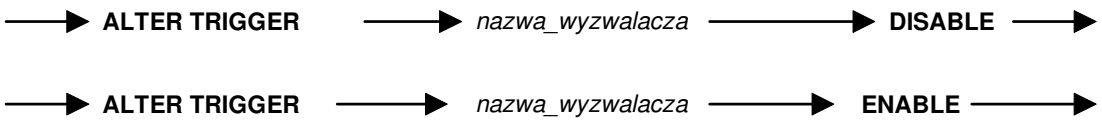
  Tylko w wyrażeniach warunkowych, występujących po ***when***, używamy odwołań do *'starych*' i '*nowych*' wartości kolumn bez poprzedzającego je dwukropka.

- jeżeli definiujemy trigger związany z kilkoma rodzajami modyfikowania danych (używając **OR**, możemy - w przypadku krańcowym - zdefiniować trigger związany z wpisywaniem, modyfikacją i usuwaniem danych - równocześnie) możemy w ciele tego triggera użyć konstrukcji postaci:

```
if inserting then
        ...
        instrukcje
        ...
elsif deleting then
        ...
        instrukcje
        ...
elsif updating then
        ...
        instrukcje
        ...
end if;
```

  Kolejność sprawdzania warunków (***inserting***, ***deleting***, ***updating***) jest dowolna, te warunki mogą wystąpić w wielu miejscach bloku PL/SQL - poza triggerem nie można użyć tych nazw warunkowych.

Triggery są pamiętane ***wyłącznie*** znakowo - nie są kompilowane. Może to doprowadzić - przy zbyt dużej ilości aktywnych triggerów - do znacznego spowolnienia przetwarzania. Triggery możemy w każdej chwili dezaktywować i aktywować - używając instrukcji:



Treść triggerów jest pamiętana w tabeli USER_TRIGGERS. Nie definiuje się praw dostępu do triggera - trigger działa niezależnie od tego, kto modyfikuje dane. Dlatego wewnątrz triggera powinniśmy odwoływać się do tabel podając nazwę użytkownika - ***scott.emp*** zamiast po prostu ***emp***.

Dodatek - definicje struktur tabel używanych w ćwiczeniach:

tabela EMP			
pole	***NULL?***	***typ***	***komentarz***
EMPNO	NOT NULL	number(4)	numer pracownika
ENAME		varchar2(10)	nazwisko pracownika
JOB		varchar2(9)	nazwa stanowiska
MGR		number(4)	numer pracownika, który jest kierownikiem bieżącego pracownika
HIREDATE		date	data zatrudnienia
SAL		number(7,2)	zarobki
COMM		number(7,2)	prowizja
DEPTNO	NOT NULL	number(2)	numer departamentu

EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

tabela DEPT - informacja o departamentach			
pole	***NULL?***	***typ***	***komentarz***
DEPTNO	NOT NULL	number(2)	numer departamentu
DNAME		varchar2(14)	nazwa departamentu
LOC		varcher2(13)	lokacja - nazwa miasta, w którym znajduje się departament

tabela SALGRADE			
pole	***NULL?***	***typ***	***komentarz***
GRADE		number	kod zaszeregowania
LOSAL		number	płaca minimalna
HISAL		number	płaca maksymalna

DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

SALGRADE

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999