

Wydawca: Stowarzyszenie Polskiej Grupy Użytkowników Systemu Oracle, Warszawa, ul. Sienna 75

*Systemy informatyczne.  
Projektowanie, implementowanie, eksploatawanie*

Materiały konferencyjne Stowarzyszenia Polskiej Grupy Użytkowników Systemu Oracle  
2008 r.

Poznań, 7-9 maja, 2008  
**VII Szkoła**  
użytkowników i deweloperów Oracle

**Oracle Database - Rozszerzenie proceduralne PL/SQL**

*Zawartość artykułów skopiowano bez redakcji  
z nośników dostarczonych przez autorów*

Wydawca: Stowarzyszenie Polskiej Grupy Użytkowników Systemu Oracle  
Maj 2008  
Nakład: 30 egzemplarzy

# ORACLE

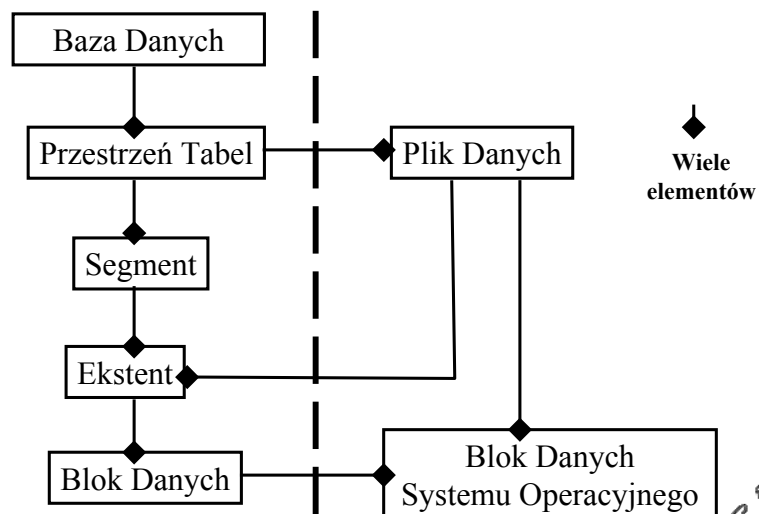
PL/SQL



## Organizacja bazy danych

**LOGICZNA**

**FIZYCZNA**





## Typy danych

Typ	Opis
VARCHAR2(rozmiar)	Ciąg znaków o zmiennej długości. Maksymalna długość : 4000 znaków , minimalna - 1 znak. Specyfikacja maksymalnej długości jest niezbędna.
NVARCHAR2(rozmiar)	Ciąg znaków o zmiennej długości. Maksymalna długość jest reprezentowana przez ilość bajtów niezbędną do reprezentacji pojedynczego znaku. Maksymalna długość : 4000 znaków. Specyfikacja maksymalnej długości jest niezbędna.
NUMBER(p,s)	Liczba mająca p miejsc całkowitych i s miejsc po przecinku
LONG	Ciąg znaków o zmiennej długości. Maksymalna długość 2 GB
DATE	Data od 1 stycznia 4712 p.n.e do 31 grudnia 9999 n.e
RAW(rozmiar)	Czyste dane o długości równej ilości bajtów. Maksymalna długość: 4000 bajtów
LONG RAW	Czyste dane o długości równej ilości bajtów. Maksymalna długość: 2 GB
ROWID	Szestnastkowy ciąg reprezentujący logiczny adres krotki zorganizowanej w indeks. Minimalny rozmiar - 1 bajt.
UROWID	Szestnastkowy ciąg reprezentujący logiczny adres krotki zorganizowanej w indeks. Maksymalny (i defaultowy) rozmiar - 4000 bajtów.
CHAR(rozmiar)	Ciąg o stałej długości. Maksymalny rozmiar - 2000 bajtów. Standardowy - 1 bajt.
NCHAR(rozmiar)	Ciąg o stałej długości. Maksymalny rozmiar określony ilością bajtów na znak - 2000 bajtów. Standardowy - 1 bajt.



## Typy danych cd

CLOB	Obiekt zawierający duże ilości tekstu (do 4 GB) gdzie jeden znak jest reprezentowany przez jeden bajt.
NCLOB	Obiekt zawierający duże ilości tekstu (do 4 GB) gdzie jeden znak jest reprezentowany przez kilka bajtów.
BLOB	Duży binarny plik o maksymalnym rozmiarze 4 GB.
BFILE	Zawiera lokalizację binarnego pliku przechowywanego na zewnątrz bazy danych. Maksymalny rozmiar 4 GB



## Znaki specjalne

Symbol	Znaczenie	Przykład
.	separator w nazwach kwalifikowanych	Tabela.pole
' lub ''	ogranicznik łańcucha	'to jest napis'
:=	przypisanie	i:=1;
	konkatenacja	'Jan'    ''    'Kowalski'
--	Komentarz pojedyncza linia	--komentarz
/* */	Komentarz wieloliniowy	/* komentarz1 Komentarz2*/

## Zmienne

Nazwa zmiennej musi zaczynać się od litery, po której może wystąpić dowolnie wiele liter cyfr lub znaków specjalnych \$, # lub \_.  
Długość nazwy zmiennej nie może przekraczać 30 znaków i nie może zawierać spacji.

```
<nazwa_zmiennej> <typ_danych> [not null] [= <wartość_pocz.>];
```

```
<nazwa stałej> constant < typ_danych > := <wartość>;
```

```
licz number (5) not null: = 1111;  
nazwa varchar2 (30);  
wzrost real (5.2) := 12.5;  
razem constant integer (2) := 30;  
dzisiaj date;  
stan boolean;  
a,b,c number;
```

```
< nazwa_zmiennej > <obiekt> %type [not null] [= <wartość_pocz.>];
```

# Struktura procedury (anonimowej) PL SQL

```
DECLARE
... Deklaracje zmiennych i stałych
BEGIN
... – Ciało procedury (część wykonawcza)
EXCEPTION
... – Sekcja wyjątków
END;
```

## Przykład

```
SET SERVEROUTPUT ON;
DECLARE
BEGIN
DBMS_OUTPUT.PUT_LINE('tekst napisu');
END;
```



## Instrukcje warunkowe IF THEN

```
If licz > 18 then licz := licz + 1;  
end if ;
```

```
If var1 > 10 then  
    if var2 < var1 then  
        var2 := var1 + 20;  
    end if;  
end if;
```



## Instrukcje warunkowe

### IF THEN ELSE

```
If var1 > 10 then
    var2 := var1 + 20;
else
    var2 := var1 * var1;
end if;
```

### IF THEN ELSEIF ELSE

```
If var1 > 10 then
    var2 := var1 + 20;
elseif var1 between 7 and 8 then
    var2 := 2 * var1;
else
    var2 := var1 * var1;
end if;
```



## Pętle

### LOOP EXIT END LOOP

```
licz := 1;           -- inicjalizacja licznika
LOOP                -- LOOP słowo kluczowe (bez średnika)
    licz := licz + 1; -- inkrementacja licznika pętli
    IF licz > 100 THEN -- sprawdzenie warunku zakończenia
        ...
        exit;         -- wyjście z pętli
    END IF;
    ...
    ...
END LOOP;           -- END LOOP słowo kluczowe
---
```



## Pętle

### WHILE LOOP END LOOP

```
cut := 1;
  WHILE cut <= 100 LOOP
    ...
    ...
    cut := cut + 1;    -- inkrementacja licznika
  END LOOP;
```

### FOR IN LOOP END LOOP

(zawsze wykonywana zdefiniowaną liczbę razy)

```
FOR licz IN 1..13
LOOP
  ...
END LOOP;
```

Niedozwolona jest inkrementacja licznika we wnętrzu pętli



## Pętle

### FOR IN REVERSE LOOP END LOOP

(zawsze wykonywana zdefiniowaną liczbę razy)

```
SET SERVEROUTPUT ON;
DECLARE
liczba VARCHAR2(5) := '5639';
dlugosc NUMBER(2);
odwrocona VARCHAR2(5);
BEGIN
dlugosc:= LENGTH(liczba);
FOR licz IN REVERSE 1.. dlugosc
LOOP
odwrocona:= odwrocona || SUBSTR(liczba, licz, 1);
END LOOP;
DBMS_OUTPUT.PUT_LINE('Liczba = ' || liczba);
DBMS_OUTPUT.PUT_LINE('Odwrocona = ' || odwrocona);
END;
```



## Różne

### Pusta instrukcja

```
IF licz >= 10 THEN  
    NULL;  
ELSE  
    ...  
END IF;
```

### Sekcja deklaracji

```
DECLARE  
    licz NUMBER;  
    start_date date := sysdate;
```



## Skok bezwarunkowy

```
GOTO etykieta;
```

```
.....
```

```
<<etykieta>>
```

```
.....
```

```
SET SERVEROUTPUT ON;  
DECLARE  
maks NUMBER;  
BEGIN  
SELECT Max(Brutto) INTO maks FROM Zarobki;  
IF maks < 7000 THEN  
GOTO Upd;  
END IF;  
DBMS_OUTPUT.PUT_LINE('Przekroczono limit');  
EXIT;  
<< Upd >>  
DBMS_OUTPUT.PUT_LINE('Nie przekroczono limitu');  
END;
```



## Procedury składowane

AAA

DROP PROCEDURE up;

CREATE OR REPLACE PROCEDURE up IS  
BEGIN  
    UPDATE osoby SET nazwisko=UPPER(nazwisko);  
END;

CREATE OR REPLACE PROCEDURE wysocy IS  
BEGIN  
    INSERT INTO wys\_tab(Nazwisko,wzrost)  
    SELECT Nazwisko, wzrost FROM Osoby  
    ORDER BY wzrost DESC;  
END wysocy;



## Procedury składowane

BBB

CREATE OR REPLACE PROCEDURE wysocy  
(mm number)  
IS  
BEGIN

    INSERT INTO wys\_tab(Nazwisko,wzrost)  
    SELECT Nazwisko, wzrost FROM Osoby  
    WHERE wzrost >mm  
    ORDER BY wzrost DESC;  
END wysocy;



## Procedury składowane

### CCCccc

```
CREATE OR REPLACE PROCEDURE Dodaj
(Num number, Dodatek number)
IS
BEGIN
INSERT INTO Dodatki
SELECT Brutto+ Dodatek FROM Zarobki WHERE
IdOsoby=Num);
END;
```



## Procedury składowane

### BBBA

```
CREATE OR REPLACE PROCEDURE wysocy
(mm number, ile OUT number)
IS
BEGIN
SELECT COUNT(wzrost) INTO ile FROM Osoby
WHERE wzrost >mm;
END wysocy;
```

```
SET SERVEROUTPUT ON;
DECLARE ile number;
begin
wysocy(1.8,ile);
DBMS_OUTPUT.PUT_LINE(ile);
end;
```



## Procedury składowane

```
CREATE OR REPLACE PROCEDURE wysocy
(mm NUMBER DEFAULT 1.7, ile OUT NUMBER)
IS
BEGIN
SELECT COUNT(wzrost) INTO ile FROM Osoby
WHERE wzrost >mm;
END wysocy;
```

Przy odwoływaniu się do wartości domyślnych tylko  
wywołanie nazewnicze, bo omijamy pierwszy parametr

```
SET SERVEROUTPUT ON;
DECLARE ile number;
begin
wysocy(1.8,ile);
DBMS_OUTPUT.PUT_LINE(ile);
end;
```

```
SET SERVEROUTPUT ON;
DECLARE ile number;
begin
wysocy(ile=>ile);
DBMS_OUTPUT.PUT_LINE(ile);
end;
```



## Procedury składowane

```
CREATE OR REPLACE PROCEDURE wysocy
(ile OUT NUMBER, mm NUMBER DEFAULT 1.7)
IS
BEGIN
SELECT COUNT(wzrost) INTO ile FROM Osoby
WHERE wzrost >mm;
END wysocy;
```

Przy odwoływaniu się do wartości domyślnych obie konwencje  
bo omijamy ostatni (ostatnie) parametr

```
SET SERVEROUTPUT ON;
DECLARE ile number;
begin
wysocy(ile);
DBMS_OUTPUT.PUT_LINE(ile);
end;
```

```
SET SERVEROUTPUT ON;
DECLARE ile number;
begin
wysocy(ile=>ile);
DBMS_OUTPUT.PUT_LINE(ile);
end;
```



## Procedury składowane

### CCC

```
CREATE OR REPLACE PROCEDURE exe_tekst
( typ varchar2)
IS
BEGIN
EXECUTE IMMEDIATE
'UPDATE osoby SET Nazwisko=' || typ || '(Nazwisko)';
END exe_tekst;
```

```
CALL exe_tekst('UPPER');
SELECT * FROM osoby;
```

```
CALL exe_tekst('INITCAP');
SELECT * FROM osoby;
```



## Procedury składowane

### DDDD

```
CREATE OR REPLACE PROCEDURE Bład
IS
BEGIN
RAISE_APPLICATION_ERROR (-20205, 'Bład programu');
END bład;
```



## Procedury składowane

EEE

```
CREATE OR REPLACE PROCEDURE exe_tekst
(typ varchar2)
IS
BEGIN
IF UPPER(typ) NOT IN('UPPER','LOWER','INITCAP')
THEN
RAISE_APPLICATION_ERROR (-20205, 'Zła funkcja');
ELSE
EXECUTE IMMEDIATE
'UPDATE osoby SET Nazwisko=' || typ || '(Nazwisko)';
END IF;
END exe_tekst;
```



## Procedury składowane

```
CREATE OR REPLACE PROCEDURE Upd1 (typ varchar2)
IS
BEGIN
IF UPPER(TYP)='L' THEN
UPDATE Osoby SET Nazwisko = LOWER (Nazwisko);
END IF;
IF UPPER (typ) = 'U' THEN
UPDATE Osoby SET Nazwisko = UPPER (Nazwisko);
END IF;

EXCEPTION
WHEN NO_DATA_FOUND THEN
Null;
WHEN OTHERS THEN
Null;
END upd1;
```



## Wyjątki w Procedurze

```
CREATE OR REPLACE PROCEDURE czy_jest
  (num IN NUMBER, status OUT NUMBER)
IS
  kto NUMBER;
BEGIN
  SELECT IdOsoby INTO kto FROM Osoby WHERE IdOsoby=num;
  IF (kto=num) THEN
    status:=1;
    DBMS_OUTPUT.PUT_LINE ('Pracownik istnieje');
  END IF;
  EXCEPTION
  WHEN NO_DATA_FOUND THEN
    status:=0;
    DBMS_OUTPUT.PUT_LINE('Pracownik nie istnieje');
  END;
```



## Wyjątki w Procedurze

```
CREATE OR REPLACE PROCEDURE czy_jest
  (num IN NUMBER, status OUT NUMBER)
IS
  kto NUMBER;
BEGIN
  SELECT IdOsoby INTO kto FROM Osoby WHERE IdOsoby=num;

  status:=1;
  DBMS_OUTPUT.PUT_LINE('Pracownik istnieje');

  EXCEPTION
  WHEN NO_DATA_FOUND OR INVALID_NUMBER
  THEN
    status:=0;
    DBMS_OUTPUT.PUT_LINE ('Pracownik nie istnieje');
  END;
```



## Wyjątki w procedurze wywołanie

```
SET SERVEROUTPUT ON;
DECLARE kto NUMBER;
        status NUMBER;
BEGIN
kto:=1;
czy_jest (kto, status);
DBMS_OUTPUT.PUT_LINE(status);
END;
czy_jest (status => status, num => kto);
```

Wywołanie pozycyjne

Wywołanie nazewnicze



## Wyjątki w Procedurze

```
CREATE OR REPLACE PROCEDURE licz (mini NUMBER,ile out INT)
IS
brakuje EXCEPTION;
BEGIN
SELECT COUNT(IdOsoby) INTO ile FROM Osoby WHERE Wzrost> mini;
IF (ile=0) THEN
RAISE brakuje;
END IF;
EXCEPTION
WHEN brakuje THEN
RAISE;
WHEN OTHERS THEN
NULL;
END;
```

Minimalna obsługa



## Wyjątki w Procedurze

```
CREATE OR REPLACE PROCEDURE licz (mini NUMBER,ile out INT)
IS
brakuje EXCEPTION;
BEGIN
SELECT COUNT(IdOsoby) INTO ile FROM Osoby WHERE Wzrost> mini;
IF (ile=0) THEN
RAISE brakuje;
END IF;
EXCEPTION
WHEN brakuje THEN
  DBMS_OUTPUT.PUT_LINE('nie ma takich');
  DBMS_OUTPUT.PUT_LINE('kod - ' || SQLCODE);
  DBMS_OUTPUT.PUT_LINE('opis - ' || SQLERRM);

WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLCODE);
END;
```



## Wyjątki w Procedurze – PRAGMA EXCEPTION\_INIT

```
CREATE OR REPLACE PROCEDURE licz (mini NUMBER,ile out INT)
IS
brakuje EXCEPTION;
PRAGMA EXCEPTION_INIT(brakuje,-13467);
BEGIN
SELECT COUNT(IdOsoby) INTO ile FROM Osoby WHERE Wzrost> mini;
IF (ile=0) THEN
RAISE brakuje;
END IF;
EXCEPTION
WHEN brakuje THEN
  DBMS_OUTPUT.PUT_LINE('nie ma takich');
  DBMS_OUTPUT.PUT_LINE('kod - ' || SQLCODE);
  DBMS_OUTPUT.PUT_LINE('opis - ' || SQLERRM);

WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLCODE);
END;
```



## Wyjątki w procedurze wywołanie

```
SET SERVEROUTPUT ON;  
DECLARE ile NUMBER;  
BEGIN  
licz(1.8, ile);  
DBMS_OUTPUT.PUT_LINE (ile);  
END;
```

Wykaz błędów

```
SHOW ERRORS;
```



## Najczęściej występujące wyjątki serwera

Oracle Terror	SQLCODE	Description
CURSOR_ALREADY_OPEN	ORA-6511 SQLCODE= -6511	You tried to OPEN a cursor that was already OPEN. You must CLOSE a cursor before you try to OPEN or re-OPEN it.
DUP_VAL_ON_INDEX	ORA-00001 SQLCODE= -1	Your INSERT or UPDATE statement attempted to store duplicate values in a column or columns in a row which is restricted by a unique index.
INVALID_CURSOR	ORA-01001 SQLCODE= -1001	You made reference to a cursor that did not exist. This usually happens when you try to FETCH from a cursor or CLOSE a cursor before that cursor is OPENed.
INVALID_NUMBER	ORA-01722 SQLCODE = -1722	PL/SQL executes a SQL statement that cannot convert a character string successfully to a number. This exception is different from the VALUE_ERROR exception, as it is raised only from within a SQL statement.
LOGIN_DENIED	ORA-01017 SQLCODE= -1017	Your program tried to log onto the Oracle RDBMS with an invalid username-password combination. This exception is usually encountered when you embed PL/SQL in a 3GL language.



## Najczęściej występujące wyjątki serwera

Oracle Error	SQLCODE	Description
NO_DATA_FOUND	ORA-01403 SQLCODE=+100	This exception is raised in three different scenarios: (1) You executed a SELECT INTO statement (implicit cursor) that returned no rows. (2) You referenced an uninitialized row in a local PL/SQL table. (3) You read past end of file with UTL_FILE package.
NOT_LOGGED_ON	ORA-01012 SQLCODE=-1012	Your program tried to execute a call to the database (usually with a DML statement) before it had logged into the Oracle RDBMS.
PROGRAM_ERROR	ORA-06501 SQLCODE=-6501	PL/SQL encounters an internal problem. The message text usually also tells you to "Contact Oracle Support."
STORAGE_ERROR	ORA-06500 SQLCODE=-6500	Your program ran out of memory or memory was in some way corrupted.
TIMEOUT_ON_RESOURCE	ORA-00051 SQLCODE=-51	A timeout occurred in the RDBMS while waiting for a resource.



## Funkcja

DROP FUNCTION Liczf ;


CREATE OR REPLACE FUNCTION Liczf (mini NUMBER)  
RETURN INT  
IS  
ile INT;  
BEGIN  
SELECT COUNT(IdOsoby) INTO ile FROM Osoby WHERE  
Wzrost > mini;  
RETURN ile;  
END;

SET SERVEROUTPUT ON;  
DECLARE ile NUMBER;  
BEGIN  
ile:=liczf(0);  
DBMS\_OUTPUT.PUT\_LINE (ile);  
END;




## Funkcja

```
CREATE OR REPLACE FUNCTION Liczf (mini NUMBER,  
  ilep OUT NUMBER) RETURN INT  
IS  
  ile INT;  
  BEGIN  
  SELECT COUNT(IdOsoby) INTO Ile FROM Osoby WHERE  
  Wzrost> mini;  
  ilep:=ile;  
  RETURN ile;  
END;  
  
SET SERVEROUTPUT ON;  
DECLARE ile NUMBER;  
  ilep number;  
  BEGIN  
  ile:=liczf(0, ilep);  
  DBMS_OUTPUT.PUT_LINE (ile);  
  DBMS_OUTPUT.PUT_LINE (ilep);  
END;
```



## Funkcja

```
CREATE OR REPLACE FUNCTION Liczf (mini NUMBER,  
  ilep IN OUT NUMBER) RETURN INT  
IS  
  ile INT;  
  BEGIN  
  SELECT COUNT(IdOsoby) INTO Ile FROM Osoby WHERE  
  Wzrost> mini;  
  ilep:=ile;  
  RETURN ile;  
END;  
  
SET SERVEROUTPUT ON;  
DECLARE ile NUMBER;  
  ilep number;  
  BEGIN  
  ilep:=15;  
  ile:=liczf(0, ilep);  
  DBMS_OUTPUT.PUT_LINE (ile);  
  DBMS_OUTPUT.PUT_LINE (ilep);  
END;
```



## Wykorzystanie funkcji użytkownika w zapytaniu

```
CREATE OR REPLACE FUNCTION razem
(kto IN Zarobki.IdOsoby%TYPE)
RETURN NUMBER
IS
    policz NUMBER;
BEGIN
    SELECT SUM(Brutto) INTO policz FROM zarobki WHERE idosoby=kto;
    RETURN policz;
END;
```

Wywołanie

```
SELECT Nazwisko, razem(IdOsoby) FROM Osoby;
```



## Wyjątki w funkcji

```
CREATE OR REPLACE FUNCTION Liczf (mini NUMBER) RETURN INT
IS
    ile INT;
    Brakuje EXCEPTION;
BEGIN
    SELECT COUNT(IdOsoby) INTO Ile FROM Osoby WHERE Wzrost>
    mini;
    IF (ile=0) THEN
        RAISE brakuje;
    END IF;
    RETURN ile;
EXCEPTION
    WHEN brakuje THEN
        DBMS_OUTPUT.PUT_LINE('nie ma takich');
        DBMS_OUTPUT.PUT_LINE('kod - ' || SQLCODE);
        DBMS_OUTPUT.PUT_LINE('opis - ' || SQLERRM);
        --RETURN ile;
    WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLCODE);
END;
```



## Wyjątki w funkcji wywołanie

```
SET SERVEROUTPUT ON;
DECLARE ile NUMBER;
BEGIN
ile:=liczf(0);
DBMS_OUTPUT.PUT_LINE (ile);
END;
```



## Procedury i funkcje wbudowane

```
VARIABLE aaa NUMBER;
EXECUTE DBMS_RANDOM.SEED
  (TO_CHAR(SYSDATE,'MM-DD-YYYY HH24:MI:SS'));
EXECUTE :aaa:=DBMS_RANDOM.RANDOM;
PRINT aaa;
```

Stara forma wywołania (pozostałość z SQL PLUS)

```
SET SERVEROUTPUT ON;
DECLARE
aaa NUMBER;
BEGIN
DBMS_RANDOM.SEED
  (TO_CHAR(SYSDATE,'MM-DD-YYYY HH24:MI:SS'));

aaa:=DBMS_RANDOM.RANDOM;
DBMS_OUTPUT.PUT_LINE(aaa);
END;
```

Nowe "porządne" wywołanie



## Procedury i funkcje wbudowane

```
SET SERVEROUTPUT ON;
DECLARE
aaa VARCHAR2(5);
BEGIN
                                Długość łańcucha - 5

aaa:=DBMS_RANDOM.STRING('U', 5);
DBMS_OUTPUT.PUT_LINE(aaa);
END;
                                Formatowanie UPPER – 'U'; LOWER – 'L'

aaa:=DBMS_RANDOM.VALUE(0,10);
                                Zakres <0 10)
```



## Procedury i funkcje wbudowane

CIEKAWA KONWERSJA LICZBA NAPIS  
(tylko po angielsku)

```
SELECT TO_CHAR(TO_DATE(12345, 'J'), 'JSP') FROM DUAL
```

```
TO_CHAR(TO_DATE(12345,'J'),'JSP')
```

```
-----
TWELVE THOUSAND THREE HUNDRED FORTY-FIVE
```



## Funkcje SQL

### Wypisanie bieżącego roku

```
SELECT  
to_number(substr(to_char(sysdate,'yyyymmdd'),1,4)) Rok  
FROM DUAL;
```



## Procedury wyzwalane

### AAAA

```
CREATE OR REPLACE TRIGGER data_n  
BEFORE INSERT  
ON Osoby  
BEGIN  
    UPDATE Osoby SET data = DBMS.DATE  
WHERE data IS NULL;  
END;
```



## Procedury wyzwalane BBB

```
CREATE OR REPLACE TRIGGER UP_TR  
AFTER INSERT  
ON Osoby  
BEGIN  
UPDATE Osoby SET Imie=UPPER(Imie);  
END;
```



## Procedury wyzwalane CCC

```
CREATE OR REPLACE TRIGGER up_tr  
BEFORE UPDATE OF Nazwisko ON Osoby  
FOR EACH ROW  
  
BEGIN  
:New.Imie:=UPPER(:New.Nazwisko);  
END;
```



## Procedury wyzwalane

### DDD

```
CREATE OR REPLACE TRIGGER sp_tr1
AFTER INSERT ON Osoby
FOR EACH ROW

BEGIN
UPDATE Osoby SET Imie=UPPER(:New.Nazwisko)
WHERE Nazwisko=:New.Nazwisko
AND Imie IS NOT NULL;

END;
```



## Procedury wyzwalane

### EEE

```
CREATE OR REPLACE TRIGGER sprawdz
BEFORE UPDATE ON zarobki
FOR EACH ROW
WHEN (New.Brutto/Old.Brutto>1.1)

BEGIN
INSERT INTO spr
VALUES(:Old.IdZarobku,:Old.IdOsoby,:Old.Brutto);
END;

UPDATE Zarobki SET Brutto=3000 WHERE IdZarobku<3;
SELECT * FROM spr;
```



## Procedury wyzwalane

### FFF

```
CREATE OR REPLACE TRIGGER nowy_kod
BEFORE INSERT ON Dzialy
  FOR EACH ROW
  DECLARE
    New_Kod number;
  BEGIN

    :New.Kod := SUBSTR(:New.Opis,1,2);
  END;
```



## Procedury wyzwalane

### GGG

```
CREATE OR REPLACE TRIGGER nowy_numer
BEFORE
INSERT ON Dzialy
  FOR EACH ROW
  DECLARE
    New_Num number;
  BEGIN
    SELECT (NVL(MAX(IdDzialu),0)+1) INTO New_Num
    FROM Dzialy;
    :New.IdDzialu := New_Num;
  END;
```



## Procedury wyzwalane

### GGG

```
DROP SEQUENCE licznik;  
/  
CREATE SEQUENCE licznik START WITH 7;  
/  
CREATE OR REPLACE TRIGGER nowy_numer BEFORE  
INSERT ON Dzialy  
FOR EACH ROW  
DECLARE  
New_Num number;  
BEGIN  
SELECT licznik.NEXTVAL INTO New_Num FROM DUAL;  
:New.IdDzialu := New_Num ;  
END;  
/
```



## Procedury wyzwalane

### HHH

```
CREATE OR REPLACE TRIGGER zaliczka  
BEFORE  
INSERT ON Osoby  
FOR EACH ROW  
DECLARE M_Num int;  
BEGIN  
SELECT (NVL(MAX(IdZarobku),0)+1) INTO M_Num  
FROM ZAROBKI;  
INSERT INTO ZAROBKI  
VALUES(M_Num, :New.IdOsoby, 100) ;  
END;
```



## Procedury wyzwalane

### HHH1

```
CREATE OR REPLACE TRIGGER zaliczka
AFTER
INSERT ON Osoby
FOR EACH ROW
  DECLARE M_Num int;
BEGIN
SELECT (NVL(MAX(IdZarobku),0)+1) INTO M_Num
FROM ZAROBKI;
INSERT INTO ZAROBKI
VALUES(M_Num, :New.IdOsoby, 100) ;
END;
```



## Triggery uzupełnienia

(tabela dla przykładów)

```
CREATE TABLE studenci
(Indeks NUMBER(6) PRIMARY KEY,
Imie VARCHAR2(17),
Nazwisko VARCHAR2(20),
Login VARCHAR2(6)
)
ALTER TRIGGER N_log DISABLE;
ALTER TRIGGER N_log ENABLE;
```

```
CREATE OR REPLACE TRIGGER N_log BEFORE
INSERT OR UPDATE OF Imie, Nazwisko
ON Studenci
FOR EACH ROW
BEGIN
:NEW.Imie:=UPPER(:NEW.Imie);
:NEW.Nazwisko:=UPPER(:NEW.Nazwisko);
:NEW.Login:=UPPER(SUBSTR(:NEW.Nazwisko,1,5))
||UPPER(SUBSTR(:NEW.Imie,1,1));
END;
```



## Triggery uzupełnienia

```
CREATE OR REPLACE TRIGGER N_log BEFORE
INSERT
OR UPDATE OF Imie, Nazwisko
ON Studenci
REFERENCING OLD AS Stary NEW AS Nowy
FOR EACH ROW BEGIN
:Nowy.Imie:=UPPER(:Nowy.Imie);
:Nowy.Nazwisko:=UPPER(:Nowy.Nazwisko);
:Nowy.Login:=UPPER(SUBSTR(:Nowy.Nazwisko,1,5))
||UPPER(SUBSTR(:Nowy.Imie,1,1));
END;
```



## Triggery uzupełnienia

```
CREATE OR REPLACE TRIGGER N_log BEFORE
INSERT
OR UPDATE OF Imie, Nazwisko ,Login
ON Studenci
REFERENCING OLD AS Stary NEW AS Nowy
FOR EACH ROW WHEN (Stary.Login IS NULL)
BEGIN
:Nowy.Imie:=UPPER(:Nowy.Imie);
:Nowy.Nazwisko:=UPPER(:Nowy.Nazwisko);
:Nowy.Login:=UPPER(SUBSTR(:Nowy.Nazwisko,1,5))
||UPPER(SUBSTR(:Nowy.Imie,1,1));
END;
```



## Triggery uzupełnienia – sprawdzenie, która akcja wyzwoliła trigger

```
CREATE OR REPLACE TRIGGER jaka_akcja
BEFORE INSERT OR UPDATE OR DELETE ON Studenci
BEGIN
IF INSERTING THEN
DBMS_OUTPUT.PUT_LINE('Wstawianie');
END IF;
IF DELETING THEN
DBMS_OUTPUT.PUT_LINE('Usuwanie');
END IF;
IF UPDATING THEN
DBMS_OUTPUT.PUT_LINE('Aktualizowanie');
END IF;
END;
```



## Triggery uzupełnienia – sprawdzenie, która akcja wyzwoliła trigger

Widok do triggera instead off

```
CREATE OR REPLACE VIEW Stud
AS
SELECT Imie, Indeks, Login, Nazwisko
FROM Studenci
WHERE Nazwisko IS NOT NULL
WITH READ ONLY
```

Zamiast opcji READ ONLY blokada zapisu z poziomu triggera

```
CREATE OR REPLACE TRIGGER st_tr
INSTEAD OF DELETE
ON Stud
BEGIN
DBMS_OUTPUT.PUT_LINE('Zabronione');
END;
```



## Triggery uzupełnienia – sprawdzenie, która akcja wyzwoliła trigger

Widok do trigera instead off

```
CREATE OR REPLACE TRIGGER st_tr
INSTEAD OF DELETE
ON Stud FOR EACH ROW
DECLARE ile NUMBER;
BEGIN
SELECT COUNT(IdOsoby) INTO ile FROM
Zarobki WHERE IdOsoby=:OLD.IdOsoby;
IF (ile=0) THEN
DELETE FROM Osoby WHERE IdOsoby=:OLD.IdOsoby;
ELSE
DBMS_OUTPUT.PUT_LINE('Zabronione');
END IF;
END;
```

```
CREATE OR REPLACE VIEW
Stud
AS
SELECT Idosoby, Imie, Nazwisko
FROM Osoby
```



## Triggery uzupełnienia – wyłączenie wszystkich triggerów dla tabeli

```
ALTER TABLE Dzialy DISABLE ALL TRIGGERS;
```

```
ALTER TABLE Dzialy ENABLE ALL TRIGGERS;
```



## Triggery uzupełnienia – trigery dla obiektów

```
CREATE OR REPLACE TRIGGER DLA_SYSTEM
BEFORE DROP
ON SCHEMA
BEGIN
RAISE_APPLICATION_ERROR (-20555, 'Próba wykasowania : ' ||
    USER || ' - ' || UID);
END dla_system;
```

ORA-00604: wystąpił błąd na poziomie 1 rekurencyjnego SQL  
ORA-20555: Próba wykasowania : AP - 136  
ORA-06512: przy linia 2



## Pokaż błędy kompilacji

```
CREATE OR REPLACE TRIGGER A_LOG BEFORE
INSERT OR UPDATE OF IMIE, NAZWISKO
ON STUDENCI
FOR EACH ROW WHEN (New.Login IS NULL)
BEGIN
:New.Imie:=UPPER(:Nowy.Imie);
END;
```

**SHOW ERRORS;**

Błędy dla TRIGGER a\_log:

LINE/COL ERROR

-----  
2/18 PLS-00049: niepoprawna zmienna wiązania 'NOWY.IMIE'

**SHOW ERRORS TRIGGER a\_log;**

Składnia wywołania:

```
SHOW ERRORS [{ FUNCTION | PROCEDURE | PACKAGE | PACKAGE BODY |
TRIGGER | VIEW | TYPE | TYPE BODY | DIMENSION | JAVA SOURCE |
JAVA CLASS } [schemat.][nazwa]
```



```

CREATE OR REPLACE TRIGGER Audit_Osoby
AFTER INSERT OR UPDATE OR DELETE ON Osoby
FOR EACH ROW
DECLARE
    Time_now DATE;
    Terminal VARCHAR2(10);
    Licz int;
BEGIN
    Time_now := SYSDATE;
    Terminal := USERENV('TERMINAL');
    SELECT max(IdAudit)+1 INTO Licz FROM Audit_table;
    IF INSERTING THEN
        INSERT INTO Audit_Table
            VALUES (Licz,User, Time_now, Terminal, 'Osoby', 'INSERT', :new.IdOsoby);

    ELSIF DELETING THEN
        INSERT INTO Audit_Table
            VALUES (Licz, User, Time_now, Terminal, 'Osoby', 'DELETE', :old.IdOsoby);

    ELSE
        INSERT INTO Audit_Table
            VALUES (Licz, User, Time_now, Terminal, 'Osoby', 'UPDATE', :old.IdOsoby);

    IF UPDATING ('RokUrodz') THEN
        INSERT INTO Audit_Table_values
            VALUES (Licz, 'Rok_urodz', :old.RokUrodz, :new.RokUrodz);

    ELSIF UPDATING ('IdDzialu') THEN
        INSERT INTO Audit_Table_values
            VALUES (Licz, 'IdDzialu', :old.IdDzialu, :new.IdDzialu);
    END IF;
END IF;
END IF;

```

## Trigger zastosowanie do auditu

```

CREATE TABLE Audit_Table (
    IdAudit NUMBER,
    Kto VARCHAR2(10),
    Time_now DATE,
    Terminal VARCHAR2(10),
    Tabela VARCHAR2(10),
    Opetracja VARCHAR2(10),
    klucz NUMBER);

```

```

CREATE TABLE
Audit_Table_values (
    IdAudit NUMBER,
    Kolumna VARCHAR2(10),
    StaraW NUMBER,
    NowaW NUMBER);

```



```

CREATE OR REPLACE TRIGGER Ap.Polacz
AFTER LOGON ON DATABASE
DECLARE
    Time_now DATE;
    Terminal VARCHAR2(20);
    Host_N VARCHAR2(20);
    Autent VARCHAR2(20);
    Licz int;
BEGIN
    Time_now := SYSDATE;
    Terminal := USERENV('TERMINAL');
    Host_N :=SYS_CONTEXT('USERENV',
    'HOST');
    Autent:=SYS_CONTEXT('USERENV',
    AUTHENTICATION_TYPE');
    SELECT max(IdAudit)+1 INTO Licz FROM
    Loginy;
    INSERT INTO Loginy
        VALUES (Licz, User, Time_now,
        Terminal,Host_N, Autent);
END;

```

## Trigger zastosowanie do auditu

```

CREATE TABLE Loginy (
    IdAudit NUMBER,
    Kto VARCHAR2(20),
    Time_now DATE,
    Terminal VARCHAR2(20),
    Host_N VARCHAR2(20),
    Autent VARCHAR2(20)
);

```



## Funkcja SYS\_CONTEXT

**SELECT SYS\_CONTEXT('USERENV', 'Atrybut') FROM DUAL;**

Atrybut	Opis
TERMINAL	Returns the operating system identifier for the client of the current session. "Virtual" in TCP/IP
LANGUAGE	Returns the language and territory currently used by the session, along with the database character set in the form: language_territory.characterset
LANG	Returns abbreviation for the language name
SESSIONID	Returns auditing session identifier
INSTANCE	Returns instance identification number of the current instance
ENTRYID	Returns available auditing entry identifier
ISDBA	Returns TRUE if you currently have the DBA role enabled and FALSE if you do not.
CLIENT_INFO	Returns up to 64 bytes of user session information that can be stored by an application using the DBMS_APPLICATION_INFO package
NLS_TERRITORY	Returns the territory of the current session
NLS_CURRENCY	Returns the currency symbol of the current session
NLS_CALENDAR	Returns NLS calendar used for dates in the current session
NLS_DATE_FORMAT	Returns the current date format of the current session
NLS_DATE_LANGUAGE	Returns language used to express dates in the current session
NLS_SORT	Indicates whether the sort base is binary or linguistic
CURRENT_USER	Returns name of user under whose privilege the current session runs. Can be different from SESSION_USER from within a stored procedure (such as an invoker-rights procedure).
CURRENT_USERID	Returns the user ID of the user under whose privilege the current session runs. Can be different from SESSION_USERID from within a stored procedure (such as an invoker-rights procedure).



## Funkcja SYS\_CONTEXT

**SELECT SYS\_CONTEXT('USERENV', 'Atrybut') FROM DUAL;**

Atrybut	Opis
SESSION_USER	Returns the database user name by which the current user is authenticated
SESSION_USERID	Returns the identifier of the database user name by which the current user is authenticated
CURRENT_SCHEMA	Returns the name of the default schema being used in the current session. This can be changed with an ALTER SESSION SET SCHEMA statement.
CURRENT_SCHEMAID	Returns the identifier of the default schema being used in the current session. This can be changed with an ALTER SESSION SET SCHEMAID statement.
PROXY_USER	Returns the name of the database user (typically middle tier) who opened the current session on behalf of SESSION_USER
PROXY_USERID	Returns identifier of the database user (typically middle tier) who opened the current session on behalf of SESSION_USER
DB_DOMAIN	Returns the domain of the database as specified in the DB_DOMAIN initialization parameter
DB_NAME	Returns the name of the database as specified in the DB_NAME initialization parameter
HOST	Returns the name of the host machine on which the database is running



## Funkcja SYS\_CONTEXT

**SELECT SYS\_CONTEXT('USERENV', 'Atrybut') FROM DUAL;**

Atrybut	Opis
OS_USER	Returns the operating system username of the client process that initiated the database session
EXTERNAL_NAME	Returns the external name of the database user
IP_ADDRESS	Returns the IP address (when available) of the machine from which the client is connected
NETWORK_PROTOCOL	Returns the protocol named in the connect string (PROTOCOL=protocol)
BG_JOB_ID	Returns the background job ID
FG_JOB_ID	Returns the foreground job ID
AUTHENTICATION_TYPE	Shows how the user was authenticated (DATABASE, OS, NETWORK, PROXY)
AUTHENTICATION_DATA	Returns the data being used to authenticate the login user. If the user has been authenticated through SSL, or if the user's certificate was proxied to the database, this includes the user's X.509 certificate
CURRENT_SQL	SQL text of the query that triggers fine-grained audit event handler. Only valid inside the event handler
CLIENT_IDENTIFIER	User-defined client identifier for the session
GLOBAL_CONTEXT_MEMORY	Amount of shared memory used by global application context, in bytes



## Zdarzenia dla triggerów typu schema

Event	When Fired?	Attribute Functions
BEFORE ALTER AFTER ALTER	When a catalog object is altered.	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_type ora_dict_obj_name ora_dict_obj_owner ora_des_encrypted_password (for ALTER USER events) ora_is_alter_column, ora_is_drop_column (for ALTER TABLE events)
BEFORE DROP AFTER DROP	When a catalog object is dropped.	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_type ora_dict_obj_name ora_dict_obj_owner
BEFORE ANALYZE AFTER ANALYZE	When an analyze statement is issued	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_name ora_dict_obj_type ora_dict_obj_owner



## Zdarzenia dla triggerów typu schema

Event	When Fired?	Attribute Functions
BEFORE ASSOCIATE STATISTICS AFTER ASSOCIATE STATISTICS	When an associate statistics statement is issued	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_name ora_dict_obj_type ora_dict_obj_owner ora_dict_obj_name_list ora_dict_obj_owner_list
BEFORE AUDIT AFTER AUDIT BEFORE NOAUDIT AFTER NOAUDIT	When an audit or noaudit statement is issued	ora_sysevent ora_login_user ora_instance_num ora_database_name
BEFORE COMMENT AFTER COMMENT	When an object is commented	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_name ora_dict_obj_type ora_dict_obj_owner
BEFORE CREATE AFTER CREATE	When a catalog object is created.	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_type ora_dict_obj_name ora_dict_obj_owner ora_is_creating_nested_table (for CREATE TABLE events)



## Zdarzenia dla triggerów typu schema

Event	When Fired?	Attribute Functions
BEFORE DDL AFTER DDL	When most SQL DDL statements are issued. Not fired for ALTER DATABASE, CREATE CONTROLFILE, CREATE DATABASE, and DDL issued through the PL/SQL procedure interface, such as creating an advanced queue.	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_name ora_dict_obj_type ora_dict_obj_owner
BEFORE DISASSOCIATE STATISTICS AFTER DISASSOCIATE STATISTICS	When a disassociate statistics statement is issued	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_name ora_dict_obj_type ora_dict_obj_owner ora_dict_obj_name_list ora_dict_obj_owner_list
BEFORE GRANT AFTER GRANT	When a grant statement is issued	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_name ora_dict_obj_type ora_dict_obj_owner ora_grantee ora_with_grant_option ora_privileges



## Zdarzenia dla triggerów typu schema

Event	When Fired?	Attribute Functions
BEFORE LOGOFF	At the start of a user logoff	ora_sysevent ora_login_user ora_instance_num ora_database_name
AFTER LOGON	After a successful logon of a user.	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_client_ip_address
BEFORE RENAME AFTER RENAME	When a rename statement is issued.	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_name ora_dict_obj_owner ora_dict_obj_type
BEFORE REVOKE AFTER REVOKE	When a revoke statement is issued	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_name ora_dict_obj_type ora_dict_obj_owner ora_revokee ora_privileges



## Zdarzenia dla triggerów typu schema

Event	When Fired?	Attribute Functions
AFTER SUSPEND	After a SQL statement is suspended because of an out-of-space condition. The trigger should correct the condition so the statement can be resumed.	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_server_error ora_is_servererror space_error_info
BEFORE TRUNCATE AFTER TRUNCATE	When an object is truncated	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_dict_obj_name ora_dict_obj_type ora_dict_obj_owner



## Zdarzenia dla triggerów typu database

Event	When Fired?	Conditions	Restrictions	Transaction	Attribute Functions
STARTUP	When the database is opened.	None allowed	No database operations allowed in the trigger. Return status ignored.	Starts a separate transaction and commits it after firing the triggers.	ora_sysevent ora_login_user ora_instance_num ora_database_name
SHUTDOWN	Just before the server starts the shutdown of an instance. This lets the cartridge shutdown completely. For abnormal instance shutdown, this event may not be fired.	None allowed	No database operations allowed in the trigger. Return status ignored.	Starts a separate transaction and commits it after firing the triggers.	ora_sysevent ora_login_user ora_instance_num ora_database_name
SERVERERROR	When the error eno occurs. If no condition is given, then this event fires when any error occurs. Does not apply to ORA-1034, ORA-1403, ORA-1422, ORA-1423, and ORA-4030 conditions, because they are not true errors or are too serious to continue processing.	ERRNO = eno	Depends on the error. Return status ignored.	Starts a separate transaction and commits it after firing the triggers.	ora_sysevent ora_login_user ora_instance_num ora_database_name ora_server_error ora_is_servererror space_error_info

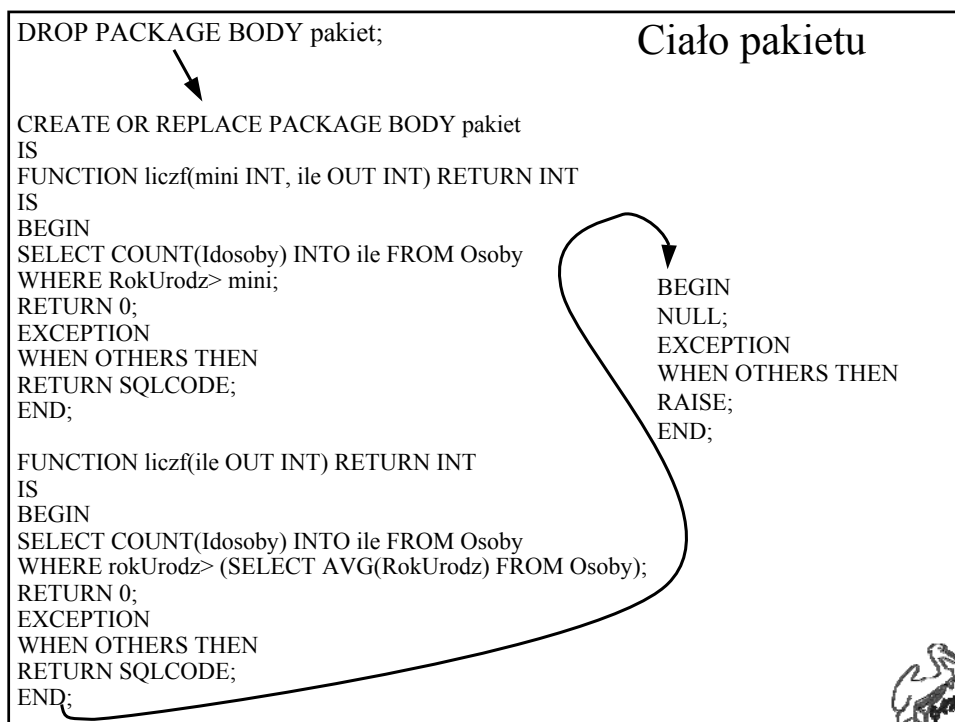


## Deklaracja pakietu

DROP PACKAGE pakiet;

CREATE OR REPLACE PACKAGE pakiet  
IS  
FUNCTION liczf(mini INT, ile OUT INT) RETURN INT;  
FUNCTION liczf(ile OUT INT) RETURN INT;  
END;





```

SET SERVEROUTPUT ON; Wywołanie pakietu
DECLARE ile INT;
blad INT;

BEGIN
blad:=pakiet.liczf(ile);
DBMS_OUTPUT.PUT_LINE(ile);
DBMS_OUTPUT.PUT_LINE(blad);

blad:=pakiet.liczf(1970,ile);
DBMS_OUTPUT.PUT_LINE(ile);
DBMS_OUTPUT.PUT_LINE(blad);

blad:=pakiet.liczf(2270,ile);
DBMS_OUTPUT.PUT_LINE(ile);
DBMS_OUTPUT.PUT_LINE(blad);

END;

```

## Deklaracja pakietu

```
DROP PACKAGE pakiet;
```

←

```
CREATE OR REPLACE PACKAGE pakiet  
IS  
  globalna INT; ← Zmienna globalna  
  FUNCTION liczf(mini INT, ile OUT INT) RETURN INT;  
  FUNCTION liczf(ile OUT INT) RETURN INT;  
  
END;
```



```
DROP PACKAGE BODY pakiet;
```

## Ciało pakietu

```
CREATE OR REPLACE PACKAGE BODY pakiet  
IS  
  globalna_wew INT;  
  FUNCTION liczf(mini INT, ile OUT INT) RETURN INT  
  IS  
  BEGIN  
    SELECT COUNT(Idosoby) INTO ile FROM Osoby  
    WHERE RokUrodz >= mini;  
    RETURN 0;  
  EXCEPTION  
  WHEN OTHERS THEN  
    RETURN SQLCODE;  
  END;  
  
  FUNCTION liczf(ile OUT INT) RETURN INT  
  IS  
  BEGIN  
    SELECT COUNT(Idosoby) INTO ile FROM Osoby  
    WHERE rokUrodz > (SELECT AVG(RokUrodz) FROM Osoby);  
    RETURN 0;  
  EXCEPTION  
  WHEN OTHERS THEN  
    RETURN SQLCODE;  
  END;
```

```
BEGIN  
  globalna := 12;  
  globalna_wew := 11;  
EXCEPTION  
WHEN OTHERS THEN  
  RAISE;  
END;
```



## Wywołanie pakietu

```
SET SERVEROUTPUT ON;  
DECLARE ile INT;  
blad INT;
```

```
BEGIN  
DBMS_OUTPUT.PUT_LINE(pakiet.globalna);  
blad:=pakiet.liczf(ile);  
DBMS_OUTPUT.PUT_LINE(ile);  
DBMS_OUTPUT.PUT_LINE(blad);  
DBMS_OUTPUT.PUT_LINE(pakiet.globalna);
```

Przy pierwszym wywołaniu NULL  
Kolejne podstawiona w pakiecie wartość

Podstawiona w pakiecie wartość aż  
do końca sesji



## Deklaracja pakietu

```
CREATE OR REPLACE PACKAGE pakiet  
IS  
FUNCTION liczf(mini INT, ile OUT INT) RETURN INT;  
FUNCTION liczf(ile OUT INT) RETURN INT;  
FUNCTION koduj(pole VARCHAR, kod CHAR) RETURN INT;  
END;
```



## Ciało pakietu - forward

```
CREATE OR REPLACE PACKAGE BODY pakiet
IS
FUNCTION liczf(mini INT, ile OUT INT)RETURN INT
...
END;

FUNCTION liczf(ile OUT INT)RETURN INT
...
END;

FUNCTION kody( kod CHAR) RETURN VARCHAR;

FUNCTION koduj(pole VARCHAR, kod CHAR) RETURN INT
IS
fun VARCHAR(11);
ttt VARCHAR(200);
BEGIN
fun:=kody(UPPER(kod));
ttt:='UPDATE Osoby SET ' || pole || '=' || fun || '(' || pole || ')';
EXECUTE IMMEDIATE ttt;
RETURN 0;
EXCEPTION
WHEN OTHERS THEN RETURN 1;
END;
```



## Ciało pakietu cd.

```
FUNCTION kody( kod CHAR) RETURN VARCHAR
IS
fun VARCHAR(11);
BEGIN
IF (kod='U') THEN
fun:='UPPER';
ELSIF (kod='L') THEN
fun:='LOWER';
ELSIF (kod='I') THEN
fun:='INITCAP';
ELSE
fun:=NULL;
END IF;
RETURN fun;
EXCEPTION
WHEN OTHERS THEN NULL;
END;
END;
```



## Wywołanie funkcji pakietu

```
SET SERVEROUTPUT ON;
DECLARE ile INT;
blad INT;

BEGIN
blad:=pakiet.koduj('nazwisko','u');
END;
```



## Kursor – podstawowy

```
DECLARE ww NUMBER;
      nn VARCHAR(15);
CURSOR sledz IS SELECT Nazwisko, RokUrodz FROM Osoby;

BEGIN
OPEN sledz;
FETCH sledz INTO nn, ww;
  WHILE SLEDZ%FOUND
  LOOP
      DBMS_OUTPUT.PUT_LINE (nn || ' ' || ww);
      FETCH sledz INTO nn, ww;
  END LOOP;
CLOSE sledz;
END;
```



## Atrybuty kursora

Atrybut	Typ	Opis
%ISOPEN	Boolean	TRUE jeśli kursor otwarty
%NOTFOUND	Boolean	TRUE jeśli ostatni FETCH nie zwrócił rekordu
%FOUND	Boolean	TRUE jeśli ostatni FETCH zwrócił rekord
%ROWCOUNT	Number	Podaje liczbę przetworzonych do tej pory rekordów



## Kursor – zmienna %ROWTYPE

```
DECLARE rec Osoby%ROWTYPE;
        CURSOR sledz IS SELECT * FROM Osoby;

BEGIN
    OPEN sledz;
    FETCH sledz INTO rec;
    WHILE sledz%FOUND
    LOOP
        DBMS_OUTPUT.PUT_LINE(SLEDZ%ROWCOUNT);
        DBMS_OUTPUT.PUT_LINE (rec.Nazwisko || ' ' || rec.RokUrodz);
        FETCH sledz INTO rec;
    END LOOP;
    CLOSE sledz;
END;
```



## Kursor – nawigacja FOR ... IN

```
SET SERVEROUTPUT ON;
DECLARE CURSOR sledz IS SELECT * FROM Osoby;

BEGIN
  FOR rec IN sledz
  LOOP
    DBMS_OUTPUT.PUT_LINE(SLEDZ%ROWCOUNT);
    DBMS_OUTPUT.PUT_LINE (rec.Nazwisko || ' ' || rec.RokUrodz);
  END LOOP;
END;
```



## Kursor – FOR UPDATE

```
SET SERVEROUTPUT ON;
DECLARE CURSOR sledz IS SELECT * FROM Osoby FOR
  UPDATE;

BEGIN
  FOR rec IN sledz
  LOOP
    DBMS_OUTPUT.PUT_LINE (rec.Nazwisko || ' ' || rec.RokUrodz);
    IF (rec.RokUrodz IS NULL) THEN
      UPDATE Osoby SET RokUrodz=0 WHERE CURRENT OF sledz;
    END IF;
    DBMS_OUTPUT.PUT_LINE (rec.Nazwisko || ' ' || rec.RokUrodz);
  END LOOP;
END;
```



## Kursor – z parametrem (wartość domyślna)

```
SET SERVEROUTPUT ON;
DECLARE CURSOR sledz (nrdzialu NUMBER :=2) IS
    SELECT * FROM Osoby WHERE IdDzialu=nrdzialu;

BEGIN
    FOR rec IN sledz
    LOOP
        DBMS_OUTPUT.PUT_LINE (rec.Nazwisko || ' ' || rec.RokUrodz);
    END LOOP;
END;
```



## Kursor – z parametrem (wartość różna od domyślnej)

```
SET SERVEROUTPUT ON;
DECLARE CURSOR sledz (nrdzialu NUMBER :=2) IS
    SELECT * FROM Osoby WHERE IdDzialu=nrdzialu;


BEGIN
    FOR rec IN sledz(3)
    LOOP
        DBMS_OUTPUT.PUT_LINE (rec.Nazwisko || ' ' || rec.RokUrodz);
    END LOOP;
END;
```



## Kursor – z parametrem (wartość różna od domyślnej - jawne podstawienie pod parametr)

```
SET SERVEROUTPUT ON;
DECLARE CURSOR sledz (nrdzialu NUMBER :=2) IS
    SELECT * FROM Osoby WHERE IdDzialu=nrdzialu;


BEGIN
    FOR rec IN sledz (nrdzialu=>3)
    LOOP
        DBMS_OUTPUT.PUT_LINE (rec.Nazwisko || ' ' || rec.RokUrodz);
    END LOOP;
END;
```



## Kursor referencyjny

```
DECLARE
    TYPE pracownicy_type IS
        REF CURSOR RETURN Osoby%ROWTYPE;
    pracownik pracownicy_type;
    pracownicy_rec Osoby%ROWTYPE;
BEGIN
    OPEN pracownik FOR SELECT * FROM Osoby;

    FETCH pracownik INTO pracownicy_rec;
    WHILE pracownik%FOUND
    LOOP
        DBMS_OUTPUT.PUT_LINE(pracownicy_rec.nazwisko);
        FETCH pracownik INTO pracownicy_rec;
    END LOOP;
    CLOSE pracownik;
END;
```



## Kursor referencyjny (bez zadeklarowanego zapytania)

```
DECLARE
  TYPE pracownicy_type IS REF CURSOR;
  pracownik pracownicy_type;
  pracownicy_rec Osoby%ROWTYPE;
BEGIN
  OPEN pracownik FOR SELECT * FROM Osoby;

  FETCH pracownik INTO pracownicy_rec;
  WHILE pracownik%FOUND
  LOOP
    DBMS_OUTPUT.PUT_LINE(pracownicy_rec.nazwisko);
    FETCH pracownik INTO pracownicy_rec;
  END LOOP;
  CLOSE pracownik;
END;
```



## Niejawny kursor

```
SET SERVEROUTPUT ON;
BEGIN
  FOR rec IN (SELECT * FROM Osoby)
  LOOP
    DBMS_OUTPUT.PUT_LINE(rec.Nazwisko || ' ' || rec.RokUrodz);
  END LOOP;
END;
```



## Ref Cursor

```
SET SERVEROUTPUT ON;
DECLARE
TYPE cur_t is REF CURSOR;
cur cur_t;
cur1 cur_t;
res osoby%ROWTYPE;
res1 osoby%ROWTYPE;
BEGIN
OPEN cur FOR SELECT * from osoby;
  FETCH cur INTO res;
  WHILE cur%FOUND
  LOOP
OPEN cur1 FOR SELECT * FROM osoby WHERE idszefa= res.idosoby;
  FETCH cur1 INTO res1;
  WHILE cur1%FOUND
  LOOP
  DBMS_OUTPUT.PUT_LINE(res.nazwisko || ' jest szefem ' || res1.nazwisko);
  FETCH cur1 INTO res1;
  END LOOP;
FETCH cur INTO res;
END LOOP;
CLOSE cur1;
CLOSE cur;
END;
```

```
KOWALSKI jest szefem KOWALIK
KOWALSKI jest szefem KOW
NOWAK jest szefem NOWIK
NOWAK jest szefem KOWALSKI
NOWAK jest szefem NOWACKI
NOWAK jest szefem JAKOW
KOWALIK jest szefem WILK
```



## Kursor w procedurze rekurencja (wewnętrzna)

```
CREATE OR REPLACE PROCEDURE inner (kto in out int, poziom in out int)
is
type cur_t is ref cursor;
cur cur_t;
res osoby%rowtype;
begin
poziom:=poziom+1;
open cur for select * from osoby where idszefa=kto;
  fetch cur into res;
  while cur%found
  loop
kto:=res.idosoby;
  DBMS_OUTPUT.PUT_LINE(to_char(poziom)||' ' || res.nazwisko);
  inner(kto, poziom);
  fetch cur into res;
  end loop;
poziom:=poziom-1;
close cur;
end;
```



## Kursor w procedurze rekurencja (zewnętrzna)

```
CREATE OR REPLACE PROCEDURE outer
is
type cur_t is ref cursor;
cur cur_t;
res osoby%rowtype;
kto int;
poziom int;
begin
poziom:=1;
open cur for select * from osoby;
  fetch cur into res;
  while cur%found
  loop
kto:=res.idosoby;
DBMS_OUTPUT.PUT_LINE(to_char(poziom)|| ' ' || res.nazwisko );
inner(kto,poziom);
  fetch cur into res;
end loop;
close cur;
end;
```

Wywołanie

```
set serveroutput on;
```

```
begin
```

```
outer;
```

```
end;
```



## Pakiet ze zdefiniowanymi typem i zmienną globalną

Typ i zmienna są widoczne poza pakietem


```
CREATE OR REPLACE PACKAGE pakiet
IS
TYPE prac_type IS REF CURSOR RETURN Osoby%ROWTYPE;
globalna INT;
FUNCTION liczf(mini INT, ile OUT INT) RETURN INT;
FUNCTION liczf(ile OUT INT) RETURN INT;
FUNCTION koduj(pole VARCHAR, kod CHAR) RETURN INT;
END;
```



## Ciało pakiet ze zdefiniowanymi zmiennymi globalnymi i lokalnymi

```
CREATE OR REPLACE PACKAGE BODY pakiet
IS
  gobal_wew INT;
  gobal_wew1 Osoby.Nazwisko%TYPE;
  gobal_wew2 Osoby%ROWTYPE;

  FUNCTION liczf(mini INT, ile OUT INT)RETURN INT
  IS
    wew INT;
  BEGIN
    ...
  END;
  ...
  END;
```




## Użycie kursora w pakiecie

```
CREATE OR REPLACE PACKAGE P_cur AS
  TYPE Osoby_T IS REF CURSOR RETURN Osoby%ROWTYPE;

  PROCEDURE Open_Osoby_c (Osoby_c IN OUT Osoby_T,
    Dzial IN INTEGER);

  PROCEDURE Fetch_Osoby_c (Osoby_c IN Osoby_T,
    Osoby_row OUT Osoby%ROWTYPE);
END P_cur;
```



## Użycie kursora w ciele pakietu

```
CREATE OR REPLACE PACKAGE BODY P_cur AS

  PROCEDURE Open_Osoby_c (Osoby_c IN OUT Osoby_T,
                          Dzial IN INTEGER) IS
  BEGIN
    OPEN Osoby_c FOR SELECT * FROM Osoby WHERE
      IdDzialu = Dzial;
  END Open_Osoby_c;

  PROCEDURE Fetch_Osoby_c (Osoby_c IN Osoby_T,
                           Osoby_row OUT Osoby%ROWTYPE) IS
  BEGIN
    FETCH Osoby_c INTO Osoby_row;
  END Fetch_Osoby_c;
END P_cur;
```



## Użycie kursora w ciele pakietu wywołanie

```
DECLARE
  Osoby_cur P_cur.Osoby_T;
  Dzial Osoby.IdDzialu%TYPE;
  Osoby_row Osoby%ROWTYPE;

BEGIN
  Dzial := 2;

  P_cur.Open_Osoby_c(Osoby_cur, Dzial);
  LOOP
    P_cur.Fetch_Osoby_c(Osoby_cur, Osoby_row);
    EXIT WHEN Osoby_cur%NOTFOUND;
    DBMS_OUTPUT.PUT(Osoby_row.Nazwisko || ' ');
    DBMS_OUTPUT.PUT_LINE(Osoby_row.RokUrodz);
  END LOOP;
END;
```



## Użycie kursora referencyjnego w pakiecie

```
CREATE OR REPLACE PACKAGE Osoby_Dzialy_data
AS
  TYPE Cur_type IS REF CURSOR;
  PROCEDURE Open_Cur(Cur IN OUT Cur_type,
                    co IN POSITIVE);
END Osoby_Dzialy_data;
```



## Użycie kursora referencyjnego w ciele pakietu

```
CREATE OR REPLACE PACKAGE BODY Osoby_Dzialy_data
AS
  PROCEDURE Open_Cur(Cur IN OUT Cur_type,
                    co IN POSITIVE) IS
  BEGIN
    IF co = 1 THEN
      OPEN Cur FOR SELECT * FROM Osoby WHERE RokUrodz
        > 1970;
    ELSIF co = 2 THEN
      OPEN Cur FOR SELECT * FROM Dzialy;
    END IF;
  END Open_Cur;
END Osoby_Dzialy_data;
```



## Użycie kursora referencyjnego w ciele pakietu wywołanie

```
DECLARE Osoby_rec Osoby%ROWTYPE;
       Dzial_rec Dzialy%ROWTYPE;
       Cur      Osoby_Dzialy_data.Cur_type;

BEGIN
  Osoby_Dzialy_data.open_cur(Cur, 1);
  Fetch Cur INTO Dzial_rec;
  DBMS_OUTPUT.PUT(Dzial_rec.Nazwa);
  DBMS_OUTPUT.PUT_LINE(' ' || Dzial_rec.kod);

EXCEPTION
  WHEN ROWTYPE_MISMATCH THEN
    DBMS_OUTPUT.PUT_LINE('Niezgodny typ rekordowy dla Działów,
nawiguję po Osobach...');
    FETCH Cur INTO Osoby_rec;
    DBMS_OUTPUT.PUT(Osoby_rec.Nazwisko);
    DBMS_OUTPUT.PUT_LINE(' ' || Osoby_rec.RokUrodz);
END;
```



## Dynamiczny SQL

```
BEGIN
EXECUTE IMMEDIATE 'UPDATE Osoby SET
Nazwisko=INITCAP(Nazwisko)';
END;
/
SELECT * FROM Osoby;
```



## Dynamiczny SQL

```
DECLARE
ktory int;
BEGIN
EXECUTE IMMEDIATE 'SELECT MAX(IdDzialu) FROM Dzialy'
INTO ktory;
DBMS_OUTPUT.PUT_LINE(ktory);

EXECUTE IMMEDIATE 'INSERT INTO Dzialy VALUES (:1, :2)'
USING ktory+1, 'Test';
COMMIT;
END;
/
SELECT * FROM Dzialy ;
```



## Dynamiczny SQL tworzymy pomocniczą procedurę

```
CREATE OR REPLACE PROCEDURE test
(ktory IN int DEFAULT 1, ile OUT int, status IN OUT int)
AS
BEGIN
SELECT COUNT(IdOsoby) INTO ile FROM Osoby
WHERE IdDzialu = ktory;
status:=0;
EXCEPTION
WHEN OTHERS THEN
status:=1;
END;
/
```



## Dynamiczny SQL wywołujemy procedurę

```
DECLARE
dzial int:=1;
stan int;
ok int:=0;
procedura varchar(222):='test';
BEGIN
EXECUTE IMMEDIATE 'BEGIN ' || procedura || '(:1, :2, :3); END;'
  USING IN dzial, OUT stan, IN OUT ok;

IF ok != 0 THEN
  DBMS_OUTPUT.PUT_LINE('błąd');
ELSE
  DBMS_OUTPUT.PUT_LINE('Liczba pracowników wynosi - ' || stan);
END IF;
END;
```



## Dynamiczny SQL tworzymy pomocniczą funkcję

```
CREATE OR REPLACE function testf
(ktory IN int DEFAULT 1, ile OUT int) RETURN int
AS
BEGIN
SELECT COUNT(IdOsoby) INTO ile FROM Osoby
  WHERE IdDzialu = ktory;
RETURN 0;
EXCEPTION
WHEN OTHERS THEN
RETURN 1;
END;/
```



## Dynamiczny SQL wywołujemy funkcję

```
DECLARE
dział int:=1;
stan int;
ok int;

funkcja varchar(222):='testf';
BEGIN
EXECUTE IMMEDIATE 'DECLARE st int; BEGIN :st:=' || funkcja || '(:1, :2);
END;' USING OUT ok, IN dział, OUT stan;

IF ok != 0 THEN
  DBMS_OUTPUT.PUT_LINE('błąd');
ELSE
  DBMS_OUTPUT.PUT_LINE('Liczba pracowników wynosi - ' || stan);
END IF;
END;
```



## Dynamiczny SQL wywołujemy procedurę z wartością domyślną

```
DECLARE
dział int:=1;
stan int;
ok int:=0;
procedura varchar(222):='testf';
BEGIN
EXECUTE IMMEDIATE 'BEGIN ' || procedura || '(ile=>:5, status=>:9); END;'
USING OUT stan, IN OUT ok;
IF ok != 0 THEN
  DBMS_OUTPUT.PUT_LINE('błąd');
ELSE
  DBMS_OUTPUT.PUT_LINE('Liczba pracowników wynosi - ' || stan);
END IF;
END;
```



## Dynamiczny SQL zastosowanie zmiennej rekordowej

```
DECLARE
TYPE rec_t is RECORD (dzial int, nazwa varchar(20));
rec rec_t;
BEGIN
EXECUTE IMMEDIATE 'SELECT * FROM Dzialy WHERE
IdDzialu= ' || 3 INTO rec;
  DBMS_OUTPUT.PUT_LINE(rec.nazwa);
END;
```



## Dynamiczny SQL zastosowanie zmiennej rekordowej – inna definicja rekordu

```
SET SERVEROUTPUT ON;
DECLARE
TYPE rec_t is RECORD (dzial Dzialy%ROWTYPE);
rec rec_t;
BEGIN
EXECUTE IMMEDIATE 'SELECT * FROM Dzialy WHERE
IdDzialu= ' || 3 INTO rec.dzial;
  DBMS_OUTPUT.PUT_LINE(rec.dzial.nazwa);
END;
```



## Dynamiczny SQL przekazywanie zmiennej do zapytania i z zapytania

```
DECLARE
nr int;
nazwa varchar2(22);
ktory int;
BEGIN
ktory:=3;
EXECUTE IMMEDIATE 'SELECT IdDzialu, nazwa FROM Dzialy
WHERE IdDzialu=:1'
INTO nr, nazwa
USING ktory;
DBMS_OUTPUT.PUT_LINE(nazwa);
END;
```



## Dynamiczny SQL przekazywanie zmiennej do zapytania i z zapytania modyfikującego

```
DECLARE
zap VARCHAR2(200);
kto int := 11;
kiedy int;
BEGIN

zap := 'UPDATE osoby SET rokurodz = 1980 WHERE idosoby = :1
RETURNING rokurodz INTO :2';
EXECUTE IMMEDIATE zap USING kto RETURNING INTO kiedy;
DBMS_OUTPUT.PUT_LINE(kiedy);

END;
```




## Wykonanie dynamicznego SQL ze zwróceniem wartości (lista tabel użytkownika i ich liczebność)

```
SET SERVEROUTPUT ON SIZE 100000;

DECLARE
v_sql VARCHAR2(1000);
v_cnt NUMBER(10);
BEGIN


FOR rec IN (Select TABLE_NAME FROM USER_TABLES)
LOOP
v_sql:= 'DECLARE vvv int; BEGIN SELECT COUNT(*) INTO :vvv FROM
' || rec.table_name || ';END;';
EXECUTE IMMEDIATE v_sql USING OUT v_cnt;

DBMS_OUTPUT.PUT_LINE(rec.table_name || '-' || v_cnt);
v_cnt:=0;
END LOOP;
END;
```



## Wykonanie dynamicznego SQL ze zwróceniem wartości

```
CREATE OR REPLACE FUNCTION Sprawdz_ciaglosc(pKolumna IN VARCHAR2,
pTabela IN VARCHAR2) RETURN VARCHAR2 IS
vLicznik NUMBER := 0;
Licznik NUMBER;
vSelect VARCHAR2(2000);
BEGIN
vSelect := 'DECLARE Licznik int ; BEGIN SELECT COUNT (DISTINCT ' ||
pKolumna || ') INTO :Licznik FROM ' || pTabela || '; END;';
EXECUTE IMMEDIATE vSelect USING OUT vLicznik;
IF vLicznik <= 4 THEN
RETURN 'NIE';
ELSE
RETURN 'TAK';
END IF;
EXCEPTION
WHEN no_data_found THEN raise_application_error(-20005,'Brak danych w
kolumnie ' || pKolumna || ' w tabeli ' || pTabela);
END;
```



## Wykonanie dynamicznego SQL ze zwróceniem wartości Wywołanie

```
SET SERVEROUTPUT ON;
DECLARE stan VARCHAR2(111);
BEGIN
stan:=Sprawdz_ciaglosc('NAZWISKO', 'OSOBY');
DBMS_OUTPUT.PUT_LINE(stan);
END;
```



```
SET SERVEROUTPUT ON;
DECLARE zap varchar2(200);
nazwa varchar2(200);
suma int;
BEGIN
nazwa:= 'testowa';
zap:='CREATE TABLE '|| nazwa || '(nr1 int, nr2 int)';
EXECUTE IMMEDIATE zap;
zap:='INSERT INTO '|| nazwa || ' VALUES (:nr1 , :nr2 )';
EXECUTE IMMEDIATE zap USING 2, 3;
zap:='INSERT INTO '|| nazwa || ' VALUES (:nr1 , :nr2 )';
EXECUTE IMMEDIATE zap USING 5, 3;
zap:='SELECT SUM(nr1) FROM ' || nazwa ;
EXECUTE IMMEDIATE zap INTO suma;
DBMS_OUTPUT.PUT_LINE(suma);
zap:='DROP TABLE ' || nazwa;
EXECUTE IMMEDIATE zap ;
zap:='COMMIT';
EXECUTE IMMEDIATE zap;
END;
```

## Dynamiczny SQL operacje na tabeli



```

CREATE OR REPLACE PROCEDURE dyn_cur( pole varchar2, tabela varchar2)
IS
TYPE rcur IS REF CURSOR;
cur rcur;
zap varchar2(2000);
wart varchar2(20);
Begin
zap:= 'SELECT ' || pole || ' FROM ' || tabela;
OPEN cur FOR ZAP;
FETCH cur INTO wart;
WHILE cur%FOUND
LOOP
Dbms_output.Put_line(wart);
FETCH CUR INTO wart;
END LOOP;
CLOSE cur;
END;

```

### Dynamiczny SQL zastosowanie cursora referencyjnego w procedurze



### Dynamiczny SQL zastosowanie cursora referencyjnego - wywołanie procedury

```

SET SERVEROUTPUT ON;
DECLARE
pole varchar2(20);
tabela varchar2(20);
BEGIN
pole:='NAZWA';
Tabela:='DZIALY';
dyn_cur(pole,tabela);
END;

```



```

DECLARE
TYPE rcur IS REF CURSOR;
cur rcur;
zap varchar2(2000);
pole varchar2(20);
tabela varchar2(20);
wart varchar2(20);
BEGIN
pole:='NAZWA';
Tabela:='DZIALY';
zap:='SELECT ' || pole || ' FROM ' || tabela;
OPEN cur FOR ZAP;
  FETCH cur INTO wart;
  WHILE cur%FOUND
  LOOP
    Dbms_output.Put_line(wart);
    FETCH CUR INTO wart;
  END LOOP;
  CLOSE cur;
END;

```

Dynamiczny SQL zastosowanie  
 cursora referencyjnego w bloku  
 anonimowym



```

DECLARE
TYPE rcur IS REF CURSOR;
TYPE kol_r IS RECORD (NAZWA VARCHAR2(255), CONSTR CHAR(1));
TYPE kol_t IS TABLE OF KOL_R INDEX BY BINARY_INTEGER;
kol kol_t;
temp kol_r;
cur rcur;
licz INT;
ile INT;
tabela varchar2(20);
zap varchar2(2000);
BEGIN
tabela:='DZIALY';
kol.DELETE;
licz:=1;

```


Dynamiczny SQL zastosowanie  
 cursora referencyjnego oraz  
 zmiennych TABLE OF



```

ZAP:= 'SELECT tc.column_name, constraint_type FROM
USER_TAB_COLUMNS TC LEFT JOIN USER_CONS_COLUMNS CC
ON tc.table_name=cc.table_name AND tc.column_name=cc.column_name
LEFT JOIN USER_CONSTRAINTS UC ON
uc.constraint_name=cc.constraint_name
WHERE tc.table_name=' || '''' || tabela || '''';
OPEN cur FOR ZAP;
FETCH cur INTO TEMP;
WHILE cur%FOUND
  LOOP
    kol(licz):=temp;
    DBMS_OUTPUT.PUT_LINE('KOLUMNA - ' || kol(licz).nazwa || ',
OGRANICZENIE - ' || kol(licz).constr );
    LICZ:=LICZ+1;
    FETCH cur INTO temp;
  END LOOP;
CLOSE cur;


```

Dynamiczny SQL zastosowanie  
 cursora referencyjnego oraz  
 zmiennych TABLE OF- cd 

```

FOR I IN kol.FIRST .. kol.LAST
  LOOP
    DBMS_OUTPUT.PUT_LINE( 'KOLUMNA - ' || kol(i).nazwa ||
    ', OGRANICZENIE - ' || kol(i).constr );
  END LOOP;
END;

```

Dynamiczny SQL zastosowanie  
 cursora referencyjnego oraz  
 zmiennych TABLE OF cd1 

## Procedury Javy

```
CREATE or REPLACE JAVA SOURCE NAMED Hello AS
public class Hello {
public static String hello() {
return "Hello World"; } };
```

```
DROP JAVA SOURCE Hello;
```



## Procedury Javy

### Stworzenie funkcji PLSQL korzystającej z Javy

```
CREATE OR REPLACE FUNCTION hello
RETURN varchar2
AS LANGUAGE JAVA
NAME 'Hello.hello () return string';
```

### Wywołanie

```
DECLARE com varchar2(333);
BEGIN
com:=hello;
DBMS_OUTPUT.PUT_LINE(com);
END;
```



CREATE OR REPLACE JAVA SOURCE NAMED Filex AS

```
import java.io.File;
public class JFile {
public static int tVal () { return 1; };
  public static int fVal () { return 0; };
public static int delete (String fileName)
{
File myFile = new File (fileName);
try
{
boolean retval = myFile.delete();
if (retval) return tVal();
  else return fVal();
}
catch (Exception e)
{
System.err.println(e.getMessage());
return 7;
}
}
```

## Procedury Javy

```
public static String dirContents (String dir)
{
try{
File myDir = new File (dir);
String[] filesList = myDir.list();
String contents = new String();
for (int i = 0; i < filesList.length; i++)
  contents = contents + ";" + filesList[i];
return contents;}
catch (Exception e)
{
System.err.println(e.getMessage());
return e.getMessage();
}
}
```



CREATE OR REPLACE PACKAGE XFILE AS

```
FUNCTION delete (file IN VARCHAR2) RETURN NUMBER;
PROCEDURE getDirContents ( dir IN VARCHAR2, files IN OUT
  VARCHAR2);
end;
```

## Procedury Javy

CREATE OR REPLACE PACKAGE BODY XFILE as

```
g_true INTEGER;
g_false INTEGER;
```

```
FUNCTION tval RETURN NUMBER
AS LANGUAGE JAVA
  NAME 'JFile.tVal () return int';
```

```
FUNCTION fval RETURN NUMBER
AS LANGUAGE JAVA
  NAME 'JFile.fVal () return int';
```



```
FUNCTION Idelete (file IN VARCHAR2) RETURN NUMBER
AS LANGUAGE JAVA
  NAME 'JFile.delete (java.lang.String) return int';
```

```
FUNCTION delete (file IN VARCHAR2) RETURN NUMBER
AS
ok number;
BEGIN
  ok:=Idelete (file);
  RETURN ok;
EXCEPTION
  WHEN OTHERS
  THEN
    dbms_output.put_line('blad' || sqlerrm);
    RETURN 0;
END;
```

```
FUNCTION dirContents (dir IN VARCHAR2)
RETURN VARCHAR2
AS LANGUAGE JAVA
  NAME 'JFile.dirContents (java.lang.String)
  return java.lang.String';
```

## Procedury Javy



## Procedury Javy

```
PROCEDURE getDirContents (
  dir IN VARCHAR2,
  files IN OUT VARCHAR2)
```

```
IS
```

```
BEGIN
  files := dirContents (dir);
END;
```

```
BEGIN
  g_true := tval;
  g_false := fval;
END;
```



## Procedury Javy

### WYWOŁANIE

```
set serveroutput on;
declare ok number;
files varchar2(2000);
begin
ok:=xfile.delete('x:\aeaa.txt');
xfile.getDirContents ('d:', files );
dbms_output.put_line('ok = ' || ok);
dbms_output.put_line('files = ' || files);
end;
```



```
CREATE OR REPLACE PROCEDURE getErrorInfo (
  errcode OUT INTEGER,  errtext OUT VARCHAR2)
```

```
IS
```

```
c_keyword CONSTANT CHAR(23) := 'java.sql.SQLException: ';
c_keyword_len CONSTANT PLS_INTEGER := 23;
v_keyword_loc PLS_INTEGER;
v_msg VARCHAR2(1000) := SQLERRM;
```

```
BEGIN
```

```
v_keyword_loc := INSTR (v_msg, c_keyword);
```

```
IF v_keyword_loc = 0
```

```
THEN
```

```
errcode := SQLCODE;
```

```
errtext := SQLERRM;
```

```
ELSE
```

```
errtext := SUBSTR (
  v_msg, v_keyword_loc + c_keyword_len);
```

```
errcode :=
```

```
SUBSTR (errtext, 4, 6 /* ORA-NNNNN */);
```

```
END IF;
```

```
END;
```

## Procedury Javy

### Rozróżnienie błędów ORACLE i Javy



### Wywołanie dla błędu SQL

```
DECLARE
test real;
BEGIN
  test :=1/0;
EXCEPTION
  WHEN OTHERS
  THEN
    DECLARE
      v_errcode INTEGER;
      v_errtext VARCHAR2(1000);
    BEGIN
      getErrorInfo (v_errcode, v_errtext);
      DBMS_OUTPUT.PUT_LINE( v_errcode || ' - ' || v_errtext);
    END;
  END;
```

### Procedury Javy

### Rozróżnienie błędów ORACLE i Javy

### Wywołanie dla błędu SQL

(nie wykrywa dzielenia przez zero!!! Ale wykrywa zły typ zmiennej zwracanej ze względu na NULL)



### Wywołanie dla błędu SQL

(nie wykrywa dzielenia przez zero!!! Ale wykrywa zły typ zmiennej zwracanej ze względu na NULL)

```
SET SERVEROUTPUT ON;
DECLARE com real;
BEGIN
  com:=dzieli(null,0);
EXCEPTION
  WHEN OTHERS
  THEN
    DECLARE
      v_errcode INTEGER;
      v_errtext VARCHAR2(1000);
    BEGIN
      getErrorInfo (v_errcode, v_errtext);
      DBMS_OUTPUT.PUT_LINE( v_errcode || ' - ' || v_errtext);
    END;
  END;
```

### Procedury Javy

### Rozróżnienie błędów ORACLE i Javy



### Stworzenie procedury Javy i jej inkapsulacja w PLSQL

```
CREATE OR REPLACE JAVA SOURCE NAMED Zmien
```

```
AS
```

```
import java.math.*;
```

```
public class Zmiana {
```

```
    public static void zamien (int[ ] a, int[ ] b)
```

```
    {
```

```
        int temp;
```

```
        temp=b[0];
```

```
        b[0]=a[0];
```

```
        a[0]=temp;
```

```
    }
```

```
};
```

Musi być tablica, aby wymiana danych była przez referencję a nie przez wartość jak w typach prostych

Procedury Javy



### Stworzenie procedury Javy i jej inkapsulacja w PLSQL

```
CREATE OR REPLACE PROCEDURE Zamien
```

```
(a in out Number, b in out Number)
```

```
AS
```

```
LANGUAGE JAVA
```

```
    NAME 'Zmiana.zamien (int[], int[])';
```

```
/
```

```
DECLARE com int;
```

```
res int;
```

```
begin
```

```
res:=2;
```

```
com:=3;
```

```
Zamien (com, res);
```

```
DBMS_OUTPUT.PUT_LINE(com);
```

```
DBMS_OUTPUT.PUT_LINE(res);
```

```
End;
```

Procedury Javy



```
CREATE OR REPLACE JAVA SOURCE NAMED Dziel
AS
```

```
public class Dzielenie {
    public static double Dziele (double a, double b)
    {
        return a/b;
    }
};
```

```
CREATE OR REPLACE JAVA SOURCE
NAMED Dziel
```

```
AS
public class Dzielenie {
    public static double Dziele (double a,
double b)
    {
        try{
            return a/b;
        }
        catch (Exception e)
        {
            System.err.println(e.getMessage());
            return 0;
        }
    }
};
```

## Procedury Javy

**Stworzenie funkcji Javy i jej  
inkapsulacja w PLSQL**



```
CREATE OR REPLACE FUNCTION dzieli (a in real, b in real)
RETURN real
AS LANGUAGE JAVA
    NAME 'Dzielenie.Dziele (double, double) return double';
/
```

```
SET SERVEROUTPUT ON;
DECLARE com real;
BEGIN
COM:=dzieli(1,2);
DBMS_OUTPUT.PUT_LINE(com);
END;
```

## Procedury Javy

**Stworzenie funkcji Javy i jej  
inkapsulacja w PLSQL**



```

CREATE OR REPLACE AND COMPILE
JAVA SOURCE NAMED LWZROST AS
import java.sql.*;
import java.io.*;
import oracle.jdbc.*;
public class LWzrost
{
public static double ilosc(double param)
{
double ile = 0;
try
{
Connection conn =
    DriverManager.getConnection("jdbc:default:connection");
String sql =
    "SELECT COUNT(wzrost) FROM Osoby WHERE wzrost > " + param;
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery(sql);
while (rset.next())
    {
        ile = (double)rset.getInt(1);
    }
rset.close();
stmt.close();
} catch (SQLException e) {System.err.println(e.getMessage());}
return ile;
}
}

```

## Klasa Javy do funkcji zliczającej osoby wyższe niż parametr



## Enkapsulacja do funkcji w PLSQL klasy zliczającej osoby wyższe niż parametr i jej wywołanie

```

CREATE OR REPLACE FUNCTION ilWzrf
(param IN REAL) RETURN real
AS LANGUAGE JAVA
NAME 'LWzrost.ilosc(double) return double';
/
SET SERVEROUTPUT ON;
DECLARE
ok int;
BEGIN
ok:=ilWzrf(1.7);
DBMS_OUTPUT.PUT_LINE(ok);
END;
/

```



```

CREATE OR REPLACE AND COMPILE JAVA SOURCE NAMED LWZROST AS
import java.sql.*;
import java.io.*;
import oracle.jdbc.*;
public class LWzrost
{
public static void ilosc(double param,double[] ile)
{
try {
Connection conn =
    DriverManager.getConnection("jdbc:default:connection");
    String sql =
        "SELECT COUNT(wzrost) FROM Osoby WHERE wzrost > " + param;
    Statement stmt = conn.createStatement();
    ResultSet rset = stmt.executeQuery(sql);
    while (rset.next())
    {
        ile[0] = (double)rset.getInt(1);
    }
    rset.close();
    stmt.close();
}
catch (SQLException e) {System.err.println(e.getMessage());}
}
}

```

Klasa Javy  
do procedury zliczającej osoby wyższe  
niż parametr



## Enkapsulacja do procedury w PLSQL klasy zliczającej osoby wyższe niż parametr i jej wywołanie

```

CREATE OR REPLACE PROCEDURE ilWzr
(param IN REAL,ile IN OUT NUMBER)
AS LANGUAGE JAVA
NAME 'LWzrost.ilosc(double,double[])';
/
SET SERVEROUTPUT ON;
DECLARE a real;
BEGIN
ilWzr(1.7,a);
dbms_output.put_line(a);
end;

```



