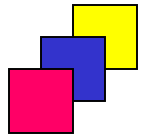
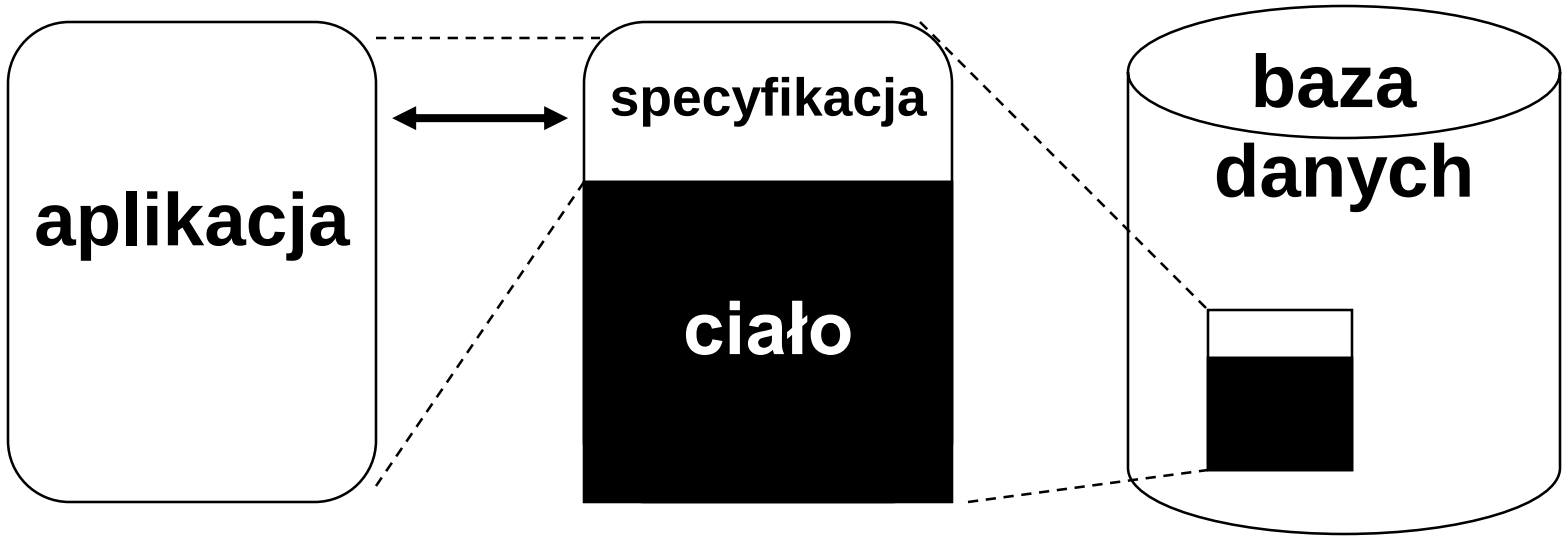
Język PL/SQL. Rozdział 5.
Pakiety podprogramów. Dynamiczny SQL

Pakiety podprogramów, specyfikacja i ciało pakietu, zmienne i kursory pakietowe, pseudoinstrukcje (dyrektywy kompilatora), dynamiczny SQL.



Pakiety

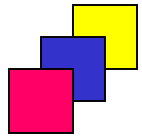
Pakiet (ang. *package*) grupuje powiązane logicznie typy, procedury, funkcje, zmienne i kursory. Składa się ze specyfikacji (interfejsu) i ciała (implementacji). W specyfikacji mieszczą się deklaracje typów, zmiennych, stałych, kursorów, wyjątków i podprogramów. W ciele mieści się implementacja specyfikacji.



Cele stosowania pakietów

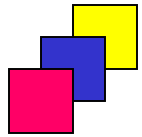
Cechy:

- **modularność**
- **ukrycie informacji – użytkownikowi jest udostępniana tylko specyfikacja pakietu (interfejs), natomiast implementacja procedur i funkcji jest niewidoczna**
- **zwiększenie funkcjonalności – zmienne zadeklarowane w pakietach istnieją przez całą sesję użytkownika; dzięki temu mogą one służyć do wzajemnej komunikacji różnych procesów**
- **zwiększenie szybkości działania – przy pierwszym odwołaniu do pakietu cała jego zawartość jest ładowana do pamięci**
- **współdzielenie przez wielu użytkowników**
- **możliwość zmiany implementacji (ciała) pakietu bez konieczności rekompilacji modułów zależnych**



Kiedy rozważać użycie pakietów

- **Gdy chcemy ukryć sposób przetwarzania danych – zamiast wykonywać operacje DML bezpośrednio na tabelach dostarczamy proceduralne API.**
  Pakiet pełni wówczas rolę interfejsu nazywanego *table API* lub *transaction API.*
- **Gdy chcemy uniknąć umieszczania tych samych stałych wewnątrz różnych modułów**
- **Gdy chcemy zgrupować logicznie powiązane ze sobą moduły programowe**
- **Gdy chcemy skorzystać z danych wyliczonych raz na poziomie sesji i dostępnych w buforze przez cały czas jej trwania**



Definiowanie pakietu

```
CREATE [OR REPLACE] PACKAGE
nazwa_pakietu IS
    deklaracje stałych, zmiennych, kursorów
    dostępnych na zewnątrz pakietu
    deklaracje procedur i funkcji
END [nazwa_pakietu];
```

specyfikacja

```
CREATE [OR REPLACE] PACKAGE BODY
nazwa_pakietu IS
    deklaracje stałych, zmiennych, kursorów
    dostępnych wewnątrz pakietu
    definicje procedur i funkcji
END [nazwa_pakietu];
```

ciało

Przykład pakietu

```
CREATE OR REPLACE PACKAGE kadry IS
        PROCEDURE nowy_pracownik(p_nazwisko VARCHAR2);
        FUNCTION podatek(p_id_prac NUMBER) RETURN NUMBER;
        bledny_pracownik EXCEPTION;
END kadry;
```

```
CREATE OR REPLACE PACKAGE BODY kadry IS
        v_liczba_pracownikow NUMBER := 0;
        PROCEDURE nowy_pracownik(...) IS
        BEGIN ...
        END nowy_pracownik;
        FUNCTION podatek(...) RETURN NUMBER IS
        BEGIN ...
        END podatek;
END kadry;
```

Wywołanie procedury lub funkcji pakietu

- **Do zdefiniowanego w specyfikacji pakietu obiektu (procedury, funkcji, zmiennej, kursora, …) odwołujemy się spoza pakietu poprzedzając jego nazwę nazwą pakietu, np.**

```
EXECUTE kadry.usun_pracownika(230)
```

```
SELECT kadry.zl_na_usd(placa_pod) FROM  pracownicy;
```

- **Odwołanie do obiektu z wnętrza jego własnego pakietu nie wymaga prefiksu**

```
CREATE OR REPLACE PACKAGE BODY kadry IS
        v_liczba_pracownikow NUMBER := 0;
        PROCEDURE nowy_pracownik(...) IS
        BEGIN
          v_liczba_pracownikow := v_liczba_pracownikow + 1;
          …
        END nowy_pracownik;
…
```

Kompilowanie pakietu

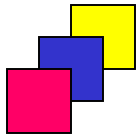
```
ALTER PACKAGE nazwa COMPILE PACKAGE | PACKAGE BODY;
```

Uwaga!

Jeśli zmiana definicji procedur lub funkcji pakietu nie zmienia ich sygnatury (nie ma konieczności rekompilacji specyfikacji, rekompilacji podlega jedynie ciało), nie ma konieczności unieważnienia (a co za tym idzie, rekompilacji) podprogramów spoza pakietu, je wywołujących.

Usuwanie pakietu

```
DROP PACKAGE | PACKAGE BODY nazwa;
```

Zmienne pakietowe

- **Deklarowane w specyfikacji (dostępne publicznie) lub ciele (dostępne tylko z podprogramów) pakietu.**
- **Zachowują nadane wartości przez całą sesję użytkownika.**

```
CREATE OR REPLACE PACKAGE xyz IS
  l_name VARCHAR(30);
  FUNCTION x(...) ;
END xyz;
…

EXECUTE xyz.l_name := 'ABC';

BEGIN
  DBMS_OUTPUT.PUT_LINE(xyz.l_name);
END;
```

Inicjalizacja pakietu

W momencie pierwszego odwołania do dowolnego elementu z pakietu zawartość całego pakietu zostaje załadowana do bufora. Zmienne i kursory mogą być inicjalizowane w części inicjalizacyjnej pakietu.

```
CREATE OR REPLACE PACKAGE BODY xyz IS
  l_name VARCHAR(30);
  l_date_of_birth DATE;
  FUNCTION x(...) RETURN ...;
...
BEGIN
  l_name := SYS_CONTEXT ('USERENV', 'SESSION_USER');
  l_date_of_birth := SYSDATE; ...
END xyz;
```

Pakiet bez ciała

Np. jako składnica stałych, używanych w systemie.

```
CREATE OR REPLACE PACKAGE stałe IS
  c_PI constant number(3,2) := 3.14;
  c_E constant number(3,2) := 2.72;
  c_EULER constant number(5,4) := 0.5772;
END stałe;
```

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(stałe.c_PI);
END;
```

Kursory pakietowe

- Kursory mogą być deklarowane zarówno w specyfikacji pakietu jak i w ciele
- Stan kursora (otwarty, zamknięty, wskaźnik na określony rekord wyniku zapytania) utrzymywany jest przez cały stan sesji
- Deklaracja kursora w specyfikacji może mieć dwie postacie:
  - pełną – deklaracja kursora wraz z przypisanym zapytaniem
  - częściową – deklaracja jedynie nagłówka kursora bez zapytania (zapytanie jest wówczas definiowane w ciele pozwalając na jego ukrycie)

```
CREATE OR REPLACE PACKAGE pakiet_z_kursorem IS
  CURSOR c_pracownicy (p_id_zesp IN pracownicy.id_zesp%TYPE)
     RETURN pracownicy%ROWTYPE;
  PROCEDURE otworz_kursor (p_id_prac pracownicy.id_zesp%TYPE);
  PROCEDURE zamknij_kursor;
END pakiet_z_kursorem;
```

Kursory pakietowe

```
CREATE OR REPLACE PACKAGE BODY pakiet_z_kursorem IS
  CURSOR c_pracownicy (p_id_zesp IN pracownicy.id_zesp%TYPE)
    RETURN pracownicy%ROWTYPE IS
    SELECT * FROM pracownicy WHERE id_zesp = p_id_zesp;
  PROCEDURE otworz_kursor (p_id_zesp pracownicy.id_zesp%TYPE) IS
  BEGIN
    IF NOT (c_pracownicy%ISOPEN) THEN
      OPEN c_pracownicy(p_id_zesp); END IF;
  END;
  PROCEDURE zamknij_kursor IS
  BEGIN
    CLOSE c_pracownicy;
  END;
END pakiet_z_kursorem;
```

```
DECLARE
  r_prac pakiet_z_kursorem.c_pracownicy%ROWTYPE;
BEGIN
  pakiet_z_kursorem.otworz_kursor(10);
  FETCH pakiet_z_kursorem.c_pracownicy
            INTO r_prac;
  IF pakiet_z_kursorem.c_pracownicy%FOUND THEN
    dbms_output.put_line(r_prac.nazwisko);
  ELSE
    pakiet_z_kursorem.zamknij_kursor;
  END IF;
END;
```

2X

```
anonymous block completed
WEGLARZ

anonymous block completed
MAREK
```

Słownik bazy danych

- **USER_OBJECTS - informacja o pakietach w schemacie użytkownika**

```
SELECT object_name, object_type, status
FROM    user_objects
WHERE  object_type IN
           ('PACKAGE', 'PACKAGE BODY');
```

- **USER_SOURCE - kod źródłowy:**
  - **specyfikacji pakietu:**

```
SELECT text  FROM user_source
WHERE name = 'PLACE'
AND type = 'PACKAGE' ORDER BY line;
```

  - **ciała pakietu:**

```
SELECT text  FROM user_source
WHERE name = 'PLACE'
AND type = 'PACKAGE BODY' ORDER BY line;
```

Model uprawnień

- **Określenie modelu uprawnień wykorzystywanego podczas wykorzystywania składowych pakietu**
  - **możliwe jest jedynie na poziomie pakietu (specyfikacji) i**
  - **obejmuje wszystkie składowe pakietu**

```
CREATE OR REPLACE PACKAGE kadry
    AUTHID CURRENT_USER
IS
        PROCEDURE nowy_pracownik(p_nazwisko VARCHAR2);
        FUNCTION podatek(p_id_prac NUMBER) RETURN NUMBER;
        bledny_pracownik EXCEPTION;
END kadry;
```

Dynamiczny SQL

Dynamiczny SQL pozwala na konstruowanie i wykonywanie poleceń, których pełna treść nie jest znana w momencie kompilacji aplikacji, lecz dopiero w trakcie wykonywania programu.

Dynamiczny SQL pozwala na:

- **konstrukcję elastycznego kodu (np. procedury operującej na tablicy przekazanej jako parametr, dynamicznego tworzenia warunków w klauzuli WHERE)**
- **konstrukcję kodu wykonywalnego w trakcie działania programu**
- **wykonywanie w bloku PL/SQL instrukcji DDL (np. CREATE TABLE) oraz DCL (GRANT, ALTER SESSION) i instrukcji sterujących sesją, zabronionych w statycznym PL/SQL**

Polecenie EXECUTE IMMEDIATE

Polecenie EXECUTE IMMEDIATE przygotowuje (parsuje) i natychmiast wykonuje polecenie SQL bądź blok PL/SQL

```
EXECUTE IMMEDIATE polecenie
[ INTO zmienna_1 [ , zmienna_2, ... ] | rekord ]
[ USING [ IN | OUT | IN OUT ] arg_wiązania_1
      [ , [ IN | OUT | IN OUT ] arg_wiązania_2 ] ... ]
[ RETURNING INTO arg_wiązania_3, ... ] ;
```

- ***polecenie*** **to ciąg znaków zawierający polecenie SQL (bez średnika na końcu)**
- ***zmienna_i*** **to zmienna, do której zostaną wczytane wyniki zapytania**
- ***arg_wiązania_i*** **to wyrażenie, którego wartość jest przekazywana do polecenia**

Przykład dynamicznego SQL (1)

```
DECLARE
  sql_stmt VARCHAR2(100);
BEGIN
  EXECUTE IMMEDIATE
    'CREATE TABLE bonus (id NUMBER, wartosc NUMBER)';

  sql_stmt :=
     'ALTER SESSION SET NLS_DATE_FORMAT="DAY MONTH YYYY" ';
  EXECUTE IMMEDIATE sql_stmt;

  EXECUTE IMMEDIATE 'declare v_podatek number; begin v_podatek:='
    || 'kadry.podatek(150); dbms_output.put_line(v_podatek); end;';
END;
```

Przykład dynamicznego SQL (2)

```
DECLARE
   sql_stmt VARCHAR2(100);
   bnd_id_zesp number(6);
   bnd_nazwa VARCHAR2(5) := 'KADRY';
   bnd_adres VARCHAR2(25) := 'SKLODOWSKIEJ-CURIE 1';
   prac_rec pracownicy%ROWTYPE;
BEGIN
   sql_stmt := 'INSERT INTO zespoly VALUES (seq_zespol.nextval, :1, :2) ' ||
      'RETURNING id_zesp INTO :3';
   EXECUTE IMMEDIATE sql_stmt
   USING bnd_nazwa, bnd_adres RETURNING INTO bnd_id_zesp;

   sql_stmt := 'SELECT * FROM pracownicy WHERE id_prac = :id';
   EXECUTE IMMEDIATE sql_stmt INTO prac_rec USING 100;
   DBMS_OUTPUT.PUT_LINE(prac_rec.nazwisko || ' - ' || prac_rec.etat);
END;
```

Przykład dynamicznego SQL (3)

```
DECLARE
  TYPE prac_tab_typ IS TABLE of pracownicy%ROWTYPE;
  prac_tablica prac_tab_typ;
  v_id_zesp zespoly.id_zesp%TYPE := &id_zesp;
BEGIN
  EXECUTE IMMEDIATE 'SELECT * FROM pracownicy WHERE id_zesp=:1'
    BULK COLLECT INTO prac_tablica USING v_id_zesp;
  FOR i IN 1..prac_tablica.last LOOP
   DBMS_OUTPUT.PUT_LINE ('Pracownik '||prac_tablica(i).nazwisko ||
       ' zarabia '||to_char(prac_tablica(i).placa_pod));
  END LOOP;
END;
```