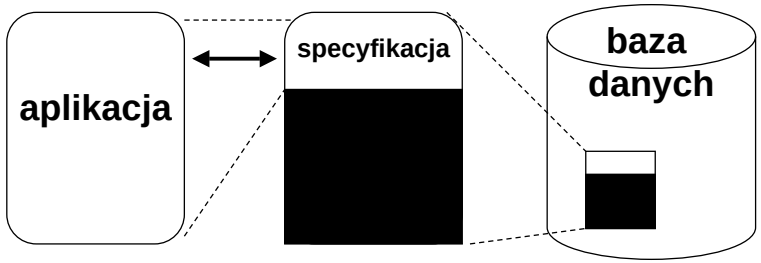
Rozdział 12
Pakiety podprogramów
Dynamiczny SQL

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 pakietu bez konieczności rekompilacji modułów zależnych**

Definiowanie pakietu

CREATE [OR REPLACE] PACKAGE nazwa_pakietu IS ← **specyfikacja**

- **deklaracje stałych, zmiennych, kursorów dostępnych na zewnątrz pakietu**
- **deklaracje procedur i funkcji**

END [nazwa_pakietu];

CREATE [OR REPLACE] PACKAGE BODY nazwa_pakietu IS ← **ciało**

- **deklaracje stałych, zmiennych, kursorów dostępnych wewnątrz pakietu**
- **definicje procedur i funkcji**

END [nazwa_pakietu];

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 nazwa_pakietu IS
  l_name VARCHAR2(30),
  l_date_of_birth DATE,
  FUNCTION x(...) RETURN ...;
...
BEGIN
  l_name := 'Kowalski'; l_date_of_birth := SYSDATE; ...
END [nazwa_pakietu];
```


Przykład pakietu

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


Wywołanie procedury lub funkcji pakietu

Zdefiniowaną w pakiecie procedurę lub funkcję wywołujemy poprzedzając jej nazwę nazwą pakietu, np.

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

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

Kompilowanie procedury, funkcji, pakietu

```
ALTER PROCEDURE | FUNCTION nazwa COMPILE;
```

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

Usuwanie procedury, funkcji, pakietu

```
DROP PROCEDURE | FUNCTION |
        PACKAGE BODY | PACKAGE nazwa;
```


Pragmy

Pragmy (pseudoinstrukcje) to dyrektywy zawierające wskazówki dla kompilatora przetwarzane na etapie kompilacji.

- EXCEPTION_INIT: pozwala na przypisanie nazwanego wyjątku do błędu Oracle
- RESTRICT_REFERENCES: potwierdza i weryfikuje czystość funkcji, wymusza sprawdzenie czystości funkcji w stosunku do informacji podanych na etapie kompilacji
- AUTONOMOUS_TRANSACTION: pozwala na wykonanie podprogramu (procedury lub funkcji) w ramach oddzielnej transakcji

Przykład pragmy

```
CREATE OR REPLACE PACKAGE place AS
  FUNCTION podatek (p_id_prac NUMBER) RETURN NUMBER;
  PRAGMA RESTRICT_REFERENCES(podatek, WNDS, RNDS, WNPS);
  deadlock_detected EXCEPTION;
END;
CREATE OR REPLACE PACKAGE BODY place AS
  FUNCTION podatek (p_id_prac NUMBER) RETURN NUMBER IS
  PRAGMA AUTONOMOUS_TRANSACTION;
  PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
    BEGIN
       -- ciało funkcji
    EXCEPTION
      WHEN deadlock_detected THEN
       -- obsługa błędu
    END podatek;
  ...
```


Słownik bazy danych

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

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

- **USER_SOURCE - kod źródłowy procedur, funkcji i pakietów użytkownika**

```
SELECT  text
FROM    user_source
WHERE   name = 'nazwa podprogramu'
ORDER BY line;
```


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);
  bnd_id_zesp NUMBER(2) := 60;
  bnd_nazwa VARCHAR2(5) := 'KADRY';
  bnd_adres VARCHAR2(25) := 'SKLODOWSKIEJ-CURIE 1';
  prac_rec pracownicy%ROWTYPE;
BEGIN
  sql_stmt := 'INSERT INTO zespoly VALUES (:1, :2, :3)';
  EXECUTE IMMEDIATE sql_stmt
  USING bnd_id_zesp, bnd_nazwa, bnd_adres;

  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 (2)

```
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;

END;
```