Język PL/SQL

1. Uwagi wstępne

Opracowanie omawia podstawowe elementy języka PL/SQL na bazie systemu ORACLE. W zamierzeniu autora ma ono stanowić materiał pomocniczy do prowadzenia wykładu oraz ćwiczeń laboratoryjnych z przedmiotu *Bazy danych.* Opracowanie może być materiałem do samodzielnego studiowania, jednak należy mieć świadomość, że brak jest w nim bardziej systematycznego omówienia języka PL/SQL. Język ten omówiono posługując się dużą liczbą przykładów, ograniczając natomiast do minimum komentarz słowny. Starano się przedstawić jedynie jego najważniejsze elementy, najbardziej przydatne w praktyce. Wiele pomniejszych kwestii jest pominiętych lub omówionych tylko pobieżnie.

Wszystkie przykłady testowane były w systemie ORACLE, w wersji 8.1.7, jednak powinny bez żadnych zmian działać również w innych wersjach – zarówno wcześniejszych jak i późniejszych.

Szczegółowy opis wszystkich poleceń języka PL/SQL w systemie ORACLE można znaleźć w dokumentacji: PL/SQL User's Guide and Reference.

Zdecydowania większość przykładów operuje na demonstracyjnym modelu o nazwie SUMMIT2, który jest standardowo instalowany w systemie ORACLE, wersja 8.x. Dlatego też należy upewnić się, że skrypt *summit2.sql* wykonał się bezbłędnie. Krótkie omówienie modelu SUMMIT2 zamieszczono na końcu niniejszego opracowania.

Wszystkie przykłady pokazane w opracowaniu zostały wykonane w programie SQL*Plus. Poniżej pokazano wygląd ekranu po prawidłowy uruchomieniu programu oraz po prawidłowym połączeniu się z jego pomocą do systemu ORACLE.

W zależności od konfiguracji systemu napisy mogą być w języku innym niż polski – domyślnie jest to oczywiście język angielski.

```
SQL*Plus: Release 8.1.7.0.0 - Production on Wto Mar 23 08:14:21 2004

(c) Copyright 2000 Oracle Corporation.  All rights reserved.

Podaj nazwę użytkownika: summit2
Podaj hasło:

Połączony z:
Oracle8i Enterprise Edition Release 8.1.7.0.0 - Production
With the Partitioning option
JServer Release 8.1.7.0.0 - Production

SQL>
```

2. Bloki anonimowe

2.1. Składnia

```
[DECLARE
   sekcja deklaracji]
BEGIN
   sekcja instrukcji
[EXCEPTION
   sekcja obsługi wyjątków]
END;
```

Bloki zagnieżdżone (ang. *nested blocks*).

```
[DECLARE
  sekcja deklaracji]
BEGIN
  sekcja instrukcji
  [BEGIN
  ...sekcja instrukcji
   [EXCEPTION
   ...sekcja obsługi wyjątków]
   END;]
[EXCEPTION
  ...sekcja obsługi wyjątków]
EBD;
```

Sekcja `DECLARE` – składnia.

```
DECLARE
  nazwa_zmiennej typ_zmiennej[długość]
  [CONSTANT]
  [:= | DEFAULT wartość_domyślna]
  [NOT NULL];
```

2.2. Przykłady

Przykład 1

Deklarowanie zmiennych i stałych – przykłady.

```
  -- Identyfikatory NIE MOGĄ zaczynać się od cyfry i być
  -- dłuższe niż 30 znaków. Identyfikatory nie mogą brzmieć
  -- tak samo jak słowa zarezerwowane (np. DATE, BEGIN)

  -- Zmienna zadeklarowana, lecz nie zainicjowana posiada wartość NULL.
  -- Poniższe trzy deklaracje są równoważne.
  licznik NUMBER(4);
  licznik NUMBER(4) := NULL;
  licznik NUMBER(4) DEFAULT NULL;
```

```
  -- Nadanie wartości zmiennym przez przypisanie.
  ilosc NUMBER(2) := 50;
  -- Można też tak:
  ilosc_2 NUMBER(2) DEFAULT 100;

  -- Literały (w tym daty) ujmujemy w apostrofy.
  imie VARCHAR2(25) DEFAULT 'Artur';
  czy_instalowac CHAR(3) := 'TAK';
  data_domyslna DATE := '01-01-2000';
```

```
  -- Wartość domyślna
  -- SYSDATE zwraca bieżącą datę oraz czas systemowy
  -- z dokładnością do jednej sekundy
  data_zatr DATE DEFAULT SYSDATE NOT NULL;
```

```
  -- Typ logiczny. Uwaga: język SQL nie posiada takiego typu.
  flaga BOOLEAN := FALSE;
```

```
  -- Stałe
  -- Poniżej zadeklarowanych stałych NIE MOŻNA zmieniać w programie.
```

```
  -- Jest to wygodny sposób na unikanie prostych błędów.
  grudzien CONSTANT NUMBER(2) := 31;
  uczelnia CONSTANT VARCHAR2(100) := 'Uniwersytet Zielonogórski';
  drobinka CONSTANT NUMBER := 0.000001;
```

Typy danych.

```
Uwaga: typy danych w PL/SQL nie odpowiadają dokładnie analogicznym typom w SQL.
-------------------------------------------------------------------------------
NUMBER        -- nazwa_zmiennej NUMBER [ ( precyzja [ ,skala ] ) ]
              -- przykłady: NUMBER, NUMBER(10), NUMBER(10,3)
INTEGER       -- Wartości całkowite, do 38 cyfr.
PLS_INTEGER   -- Wydajniejsza niż INTEGER i NUMBER,
              -- od -2.147.483.647 do 2.147.483.647
-------------------------------------------------------------------------------
VARCHAR2      -- Przechowuje ciągi o zmiennej długości od 1 do 32767 bajtów.
              -- Uwaga: w tabelach ten typ może mieć maksymalnie 4000 bajtów.
              -- Przechowywanie dłuższych ciągów wymaga typu CLOB.

VARCHAR       -- Nie zaleca się stosować. Używać raczej VARCHAR2.
CHAR          -- Typ o stałej długości. Niedobór uzupełniany spacjami.
-------------------------------------------------------------------------------
DATE          -- Data i godzina, dokładność do 1 sek.
-------------------------------------------------------------------------------
BOOLEAN       -- Może przyjmować wartość TRUE, FALSE, NULL.
-------------------------------------------------------------------------------
RECORD
TABLE
VARRAY
-------------------------------------------------------------------------------
REF CURSOR
-------------------------------------------------------------------------------
BFILE
BLOB
CLOB
NCLOB
```

2.3. Najprostsze programy

Przykład 2

Program "nic-nie-robiący".

```
BEGIN
  null;
END;
```

Przykład 3

Program "Hello, world".

```
BEGIN
  -- To jest komentarz.
  DBMS_OUTPUT.PUT_LINE('Hello, world');
  /*
  To też jest komentarz. Jest on bardziej
  przydatny, gdy trzeba skomentować
  wiele linii tekstu.
  */
END;
```

2.4. Uruchamianie

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('Hello, world');
END;
```

Poniżej pokazano przykładową sesję przy konsoli SQL*Plus-a. Pismem pochyłym podano komentarze autora.

```
SQL> connect summit2/summit2
Connected

Wpisujemy poszczególne linie kodu. Program SQL*Plus numeruje każdą wprowadzoną
linię. Po wpisaniu znaku ukośnika następuje uruchomienie wpisanego w ten sposób
kodu.
SQL> BEGIN
  2    DBMS_OUTPUT.PUT_LINE('Hello, world');
  3  END;
  4  /

PL/SQL procedure successfully completed.

Nie widzimy wyniku, gdyż zmienna SERVEROUTPUT programu SQL*Plus jest ustawiona
na OFF (jest to ustawienie domyślne). Aby widzieć wyniki wypisywane przez
DBMS_OUTPUT.PUT_LINE musi ona przyjąć wartość ON.
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2    DBMS_OUTPUT.PUT_LINE('Hello, world');
  3  END;
  4  /
Hello, world

PL/SQL procedure successfully completed.

SQL>
```

```
Przy wpisywanie dłuższych programów wygodniej jest ich kod zapisywać do pliku
tekstowego i uruchamiać wydając polecenie START lub @ (jak poniżej). Oba
polecenia są prawie równoważne sobie (szczegóły patrz dokumentacja do programu
SQL*Plus).

SQL> @c:\temp\hello.sql
Input truncated to 1 characters
Hello, world

PL/SQL procedure successfully completed.

SQL>

SQL> START c:\temp\hello.sql
Input truncated to 1 characters
Hello, world

PL/SQL procedure successfully completed.

SQL>
```

```
W powyższych przykładach pojawia się komunikat „Input truncated to 1
characters”. Aby się go pozbyć wystarczy w pliku hello.sql dodać jedną pustą
linię (po znaku ukośnika).

SQL> @c:\temp\hello.sql
Hello, world
```

```
PL/SQL procedure successfully completed.

SQL>
```

```
Wpisanie znaku ukośnika powoduje wykonanie ostatniego bloku PL?SQL, które jest
przechowywane w buforze.
SQL> /
Hello, world

PL/SQL procedure successfully completed.

SQL>

Polecenie run działa podobnie do / (ukośnik), z tą różnicą, że jest listowane
wykonywane polecenie.
SQL> RUN
  1  BEGIN
  2    DBMS_OUTPUT.PUT_LINE('Hello, world');
  3* END;
Hello, world

PL/SQL procedure successfully completed.

SQL>

Polecenie execute pozwala wykonać pojedyncze polecenie PL/SQL. Pełną nazwę
polecenia można skrócić do EXEC.
SQL> EXECUTE DBMS_OUTPUT.PUT_LINE('Hello, world');
Hello, world

PL/SQL procedure successfully completed.

SQL>
```

```
Trzeba uważać, na zmienną linesize (bo mogą pojawić się mylące wyniki).
SQL> SET LINESIZE 5
SQL> EXECUTE DBMS_OUTPUT.PUT_LINE('Hello, world')
Hello
,
world

PL/SQL procedure successfully completed.

SQL>
```

```
Polecenie list przytacza zawartość bufora. Wydane z parametrem lub parametrami
wyświetla tylko wskazaną linię lub zakres linii. Gwiazdka wskazuje linię
bieżącą.
SQL> LIST
  1  BEGIN
  2    DBMS_OUTPUT.PUT_LINE('Hello, world');
  3* END;
SQL>

SQL> LIST 2
  2*   DBMS_OUTPUT.PUT_LINE('Hello, world');

SQL> LIST 1
  1* BEGIN
SQL>
```

```
SQL> LSIT 2 3
  2    DBMS_OUTPUT.PUT_LINE('Hello, world');
  3* END;
SQL>
```

```
Kod programu zawiera błąd składniowy (usunięto średnik kończący polecenie w
drugiej linii).
SQL> @c:\temp\hello.sql
END;
*
ERROR at line 3:
ORA-06550: line 3, column 1:
PLS-00103: Encountered the symbol "END" when expecting one of the following:
:= . ( % ;
The symbol ";" was substituted for "END" to continue.

SQL>
```

3. Instrukcje warunkowe, instrukcja skoku i pętle

3.1. IF – THEN – ELSE

Przykład 4

```
DECLARE
  i NUMBER := 3;
BEGIN
  IF i > 2 THEN
    DBMS_OUTPUT.PUT_LINE('Wartość zmiennej większa od 2');
  ELSIF i < 2 THEN
    DBMS_OUTPUT.PUT_LINE('Wartość zmiennej mniejsza od 2');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Wartość zmiennej wynosi: '||i);
  END IF;
END;
```

3.2. LOOP

Przykład 5

```
DECLARE
  i NUMBER := 0;
BEGIN
  -- Pamiętaj, aby w sekwencji znalazło się polecenie, które pozwoli
  -- opuścić pętlę. Inaczej grozi zapętleniem!
  LOOP
    i := i + 1;
    DBMS_OUTPUT.PUT_LINE('Wartość zmiennej wynosi: '||TO_CHAR(i));
    EXIT WHEN i = 5;
  END LOOP;
END;
```

Przykład 6

```
DECLARE
  i NUMBER := 0;
BEGIN
  LOOP
```

```
    i := i + 1;
    DBMS_OUTPUT.PUT_LINE('Wartość zmiennej wynosi: '||TO_CHAR(i));
    IF i = 5 THEN
      EXIT;
    END IF;
  END LOOP;
END;
```

3.3. FOR – LOOP

Przykład 7

```
BEGIN
  -- Zmienna iterująca NIE MUSI być wcześniej deklarowana ani inicjowana.
  FOR i IN 1..5 LOOP
    DBMS_OUTPUT.PUT_LINE('Wartość zmiennej wynosi: '||TO_CHAR(i));
  END LOOP;

  -- Uwaga na słowo kluczowe REVERSE. Odwraca ono kierunek iteracji.
  FOR i IN REVERSE 1..5 LOOP
    DBMS_OUTPUT.PUT_LINE('Wartość zmiennej wynosi: '||TO_CHAR(i));
  END LOOP;

  -- Do wcześniejszego wyjścia z pętli można użyć polecenia EXIT
  FOR i IN 1..5 LOOP
    DBMS_OUTPUT.PUT_LINE('Wartość zmiennej wynosi: '||TO_CHAR(i));
    EXIT WHEN i = 3;
  END LOOP;

  -- Pętla nie wykona się ani razu.
  FOR i IN 6..5 LOOP
    DBMS_OUTPUT.PUT_LINE('Wartość zmiennej wynosi: '||TO_CHAR(i));
  END LOOP;
END;
```

3.4. WHILE – LOOP

Przykład 8

```
DECLARE
  i NUMBER := 1;
BEGIN
  WHILE i < 5 LOOP
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(i)||' jest nadal mniejsze niż 5');
    i := i + 1;
  END LOOP;
END;
```

3.5. GO TO

Przykład 9

```
DECLARE
  tekst VARCHAR2(100);
BEGIN
  -- Etykieta musi poprzedzać polecenie wykonywane.
  -- GOTO nie może przeskakiwać warunkowych części poleceń
  -- IF-THEN-ELSE, CASE, do polecenia LOOP i do bloku podrzędnego.
<<pierwszy>>
  tekst := 'Ala ';
  GOTO drugi;
```

```
<<wyswietl>>
  DBMS_OUTPUT.PUT_LINE(tekst);
  GOTO koniec;

<<drugi>>
  tekst := tekst||'ma ';
  GOTO trzeci;

<<trzeci>>
  tekst := tekst||'kota.';
  GOTO wyswietl;

<<koniec>>
  -- Polecenie NULL nie wykonuje ¿adnej akcji.
  NULL;
END;
```

4. Kursory

4.1. Składnia

Deklarowanie kursora.

```
CURSOR nazwa [ ( param1 typ1 [,param2 typ2] ... ) ]
[RETURN typ zwracany]
IS zapytanie SQL
```

4.2. Kursory niejawne

Przykład 10

Kursor niejawny – pobieranie danych z jednego wiersza.

```
DECLARE
  uv_nazwisko VARCHAR2(25);
  uv_imie VARCHAR2(25);
BEGIN
  SELECT first_name, last_name
  INTO uv_imie, uv_nazwisko
  FROM emp
  WHERE id = 25;
  DBMS_OUTPUT.PUT_LINE('imie: '||uv_imie||', nazwisko: '||uv_nazwisko);
END;
```

Komentarz:

Warunek, aby poniższe zadziałało: istnieje **DOKŁADNIE** jeden wiersz pasujący do wyrażenia WHERE. Gdy usuniemy np. linię `WHERE id = 25`, to otrzymamy błąd: **`ORA-01422: exact fetch returns more than requested number of rows`**. Gdyby natomiast zapytanie nie zwróciło **ŻADNEGO** rekordu, to otrzymamy błąd: **`ORA-01403 no data found`**.

Przykład 11

Kursor niejawny – pobieranie danych z jednego wiersza, deklaracje zakotwiczone.

```
DECLARE
uv_nazwisko emp.last_name%TYPE;
  uv_imie emp.first_name%TYPE;
```

```
BEGIN
  SELECT first_name, last_name
  INTO uv_imie, uv_nazwisko
  FROM emp
  WHERE id = 25;
  DBMS_OUTPUT.PUT_LINE('imie: '||uv_imie||', nazwisko: '||uv_nazwisko);
END;
```

Komentarz:

Deklarujemy zmienną o tym samym typie danych, co kolumna tabeli. Inaczej: ZAKOTWICZAMY (ang. *anchor*) lokalna deklarację do kolumny w bazie. Jeśli typ danych lub długość kolumny w bazie ulegnie zmianie, program NAJCZĘŚCIEJ automatycznie dostosuje się do nowego typu danych (ale np. z `NUMBER` na `DATE` już nie zadziała)

Przykład 12

Kursor niejawny – pobieranie danych z jednego wiersza, obsługa błędów.

```
DECLARE
  uv_nazwisko VARCHAR2(25);
  uv_imie VARCHAR2(25);
  licz PLS_INTEGER;
BEGIN
  SELECT first_name, last_name
  INTO uv_imie, uv_nazwisko
  FROM emp
  WHERE id = 25;
  DBMS_OUTPUT.PUT_LINE('imie: '||uv_imie||', nazwisko: '||uv_nazwisko);
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Zapytanie nie zwróciło danych.');
  WHEN TOO_MANY_ROWS THEN
    SELECT COUNT(*) INTO licz FROM emp;
    DBMS_OUTPUT.PUT_LINE('Zapytanie zwróciło '||licz||' rekordów.');
END;
```

Przykład 13

Atrybuty kursora niejawnego.

```
BEGIN
  -- Zmieniamy rekordy. Robimy ' salary = salary', czyli de facto
  -- zmieniamy tylko status rekordów, bez zmiany ich wartości.

UPDATE emp SET salary = salary
  WHERE salary > 1000;
  IF SQL%FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Zmieniono '||SQL%ROWCOUNT||' rekordów.');
  ELSIF SQL%NOTFOUND THEN
    DBMS_OUTPUT.PUT_LINE('Nie zmieniono żadnych rekordów.');
  END IF;
  -- Wycofujemy ew. zmiany.
  ROLLBACK;
END;
```

Komentarz:

Każde polecenie DML (`INSERT, UPDATE, DELETE, SELECT FOR UPDATE`) powoduje utworzenie kursora niejawnego (ang. *implicit cursor*). Dla takiego kursora można sprawdzać wartości atrybutów `SQL%ROWCOUNT, SQL%FOUND SQL%NOTFOUND, SQL%ISOPEN`.

4.3. Kursory jawne

Przykład 14

Kursor jawny – pobieranie danych jednego wiersza.

```
DECLARE
  uv_nazwisko VARCHAR2(25);
  uv_imie VARCHAR2(25);
  licz PLS_INTEGER;

  -- Krok 1: Deklaracja kursora.
  CURSOR c_emp IS
  SELECT first_name, last_name
  FROM emp;

BEGIN

  -- Krok 2: Otwarcie kursora.
  OPEN c_emp;

  -- Krok 3: Pobieranie danych. W tym przykładzie pobieramy tylko jeden wiersz.
  FETCH c_emp INTO uv_imie, uv_nazwisko;

  -- Krok 4. Zamknięcie kursora.
  CLOSE c_emp;

  DBMS_OUTPUT.PUT_LINE('imie: '||uv_imie||', nazwisko: '||uv_nazwisko);
END;
```

Przykład 15

Kursor jawny – pobieranie danych z więcej niż jednego wiersza w pętli `LOOP`.

```
DECLARE
  uv_nazwisko VARCHAR2(25);
  uv_imie VARCHAR2(25);
  licz PLS_INTEGER;

  -- Deklaracja kursora. Dane sortujemy wg. nazwisk.
  CURSOR c_emp IS
  SELECT first_name, last_name
  FROM emp
  ORDER BY last_name;

BEGIN

  -- Otwarcie musi być przed pętlą LOOP.
  OPEN c_emp;

  -- Pobieramy wszystkie rekordy zdefiniowane kursorem.
  LOOP
    -- Kolejne pobranie nadpisuje wartości z poprzedniego pobrania.
    -- Nie można ponownie pobrać wiersza, który już został pobrany.
    -- Po pobraniu wszystkich pasujących wierszy kolejne FETCH-e nic nie robią.
    FETCH c_emp INTO uv_imie, uv_nazwisko;
    -- Atrybut %NOTFOUND przyjmuje wartość TRUE, gdy ostatnie pobranie
    -- danych z tabeli zakończyło się niepowodzeniem. Istnieje też
    -- atrybut %FOUND.
    -- W razie pominięcia EXIT program zapętli się.
    EXIT WHEN c_emp%NOTFOUND;
    -- Atrybut %ROWCOUNT - liczba dotychczas pobranych wierszy.
    -- Całkowitą liczbę pobranych wierszy poznać można dopiero po
```

```
    -- wykonaniu ostatniego pobrania, ale przed zamknięciem kursora.
    DBMS_OUTPUT.PUT_LINE(c_emp%ROWCOUNT||'. '||uv_imie||' '||uv_nazwisko);
  END LOOP;

  -- Zamknięcie musi być po pętli LOOP.
  CLOSE c_emp;
END;
```

Przykład 16

Rekordowe struktury danych.

```
DECLARE
  -- Deklaracja kursora. Dane sortujemy wg. nazwisk.
  -- Teraz można zrobić 'SELECT *' i niżej pobierać tylko te
  -- kolumny, które chcemy.
  CURSOR c_emp IS
  SELECT *
  FROM emp
  ORDER BY last_name;
  -- Deklarujemy rekord oparty na kursorze.
  uv_emp c_emp%ROWTYPE;
BEGIN
  OPEN c_emp;
  LOOP
    FETCH c_emp INTO uv_emp;
    EXIT WHEN c_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(c_emp%ROWCOUNT||'. '||
      uv_emp.first_name||' '||uv_emp.last_name||', '||uv_emp.title);
  END LOOP;
  CLOSE c_emp;
END;
```

Przykład 17

Kursor jawny – pobieranie danych z więcej niż jednego wiersza w pętli `FOR`.

```
DECLARE
  CURSOR c_emp IS
  SELECT *
  FROM emp
  ORDER BY last_name;
BEGIN
FOR uv_emp IN c_emp
  LOOP
    DBMS_OUTPUT.PUT_LINE(uv_emp.first_name||' '||uv_emp.last_name);
  END LOOP;
END;
```

Komentarz:

Zmienna użyta w pętli FOR nie musi być zadeklarowana. PL/SQL niejawnie deklaruje zmienną, której zasięg jest ograniczony do instrukcji pętli. Można więc pominąć jawne deklarowanie zmiennej rekordowej. Można też POMINĄĆ instrukcje `OPEN`, `CLOSE` i `FETCH`.

Kursor jawny – pomijanie deklaracji kursora.

```
BEGIN
FOR uv_emp IN (
    SELECT *
    FROM emp
```

```
    ORDER BY last_name)
  LOOP
    DBMS_OUTPUT.PUT_LINE(uv_emp.first_name||' '||uv_emp.last_name);
  END LOOP;
END;
```

Komentarz:

W tej konstrukcji można pominąć deklarację kursora oraz instrukcje `OPEN`, `FETCH i CLOSE`. UWAGA: taka zwięzłość kodu nie zawsze jest pożądana. Przykładowo nie można ponownie wykorzystywać kursora. Czasami lepiej mieć wszystkie instrukcje `SELECT` w sekcji `DECLARE` – przy dłuższych programach ułatwia to jego analizowanie.

Przykład 18

Masowe wiązania (ang. *bulk binds*).

```
DECLARE
  TYPE name_t IS TABLE OF emp.last_name%TYPE; -- deklaracja typu kolekcji
  name_lista name_t;                           -- deklaracja kolekcji

  CURSOR c_emp IS SELECT last_name FROM emp;
BEGIN
  OPEN c_emp;
  FETCH c_emp BULK COLLECT INTO name_lista;    -- Brak pętli. Wszystkie dane
                                               -- pobieramy w jednej instrukcji
  CLOSE c_emp;
FOR m IN 1 .. name_lista.COUNT
LOOP
  DBMS_OUTPUT.PUT_LINE(name_lista(m));
END LOOP;
END;
```

Komentarz:

Użyliśmy tzw. *kolekcji* (czyli zmiennej pozwalającej przechowywać całą grupę elementów zamiast wartości skalarnych). Dzięki wiązaniom masowym mamy możliwość przesyłania dużych ilości danych w jednym etapie – bez konieczności pobierania ich w pętli rekord po rekordzie. Używanie wiązań masowych to często metoda na zwiększenie szybkości działania aplikacji.

4.4. Kursory z parametrami

Przykład 19

```
DECLARE
  -- Deklarujemy kursor z dwoma parametrami
  CURSOR c_emp (in_sal NUMBER, in_title VARCHAR2) IS
  SELECT *
  FROM emp
  WHERE salary > in_sal AND
        UPPER(title) = UPPER(in_title) -- odczulamy na wielkość liter
  ORDER BY salary DESC;
  uv_emp c_emp%ROWTYPE;
BEGIN
  -- Otwarcie kursora. Wielkość liter w stringu bez znaczenia.
  OPEN c_emp(1000,'StOcK CLErk' );
  LOOP
    FETCH c_emp INTO uv_emp;
    EXIT WHEN c_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(c_emp%ROWCOUNT||'. '||
      uv_emp.first_name||' '||uv_emp.last_name||', '||uv_emp.salary);
  END LOOP;
  CLOSE c_emp;
```

```
END;
```

5. Obsługa wyjątków

5.1. Przykłady

Przykład 20

```
CREATE OR REPLACE PROCEDURE testuj_wyjatki (
  in_salary IN NUMBER, in_dept_id NUMBER,
  in_licz_1 IN NUMBER, in_licz_2 IN NUMBER, in_tekst VARCHAR2) AS

  CURSOR c_emp IS
  SELECT *
  FROM emp
  WHERE salary > in_salary
  ORDER BY last_name;

  uv_emp c_emp%ROWTYPE;
  uv_last_name emp.last_name%TYPE;
  uv_tekst VARCHAR2(1);
  uv_licz PLS_INTEGER;

  -- Definiujemy nasz własny wyjątek
  ex_nasz_wyjatek EXCEPTION;

BEGIN
  -- Musi zwrócić dokładnie 1 wiersz - inaczej wyjątek TOO_MANY_ROWS
  -- lub NO_DATA_FOUND.
  SELECT last_name INTO uv_last_name FROM emp WHERE dept_id = in_dept_id;

  -- Prawdopodobieństwo wystąpienia wyjątku ZERO_DIVIDE
  SELECT in_licz_1 / in_licz_2 INTO uv_licz FROM dual;

  -- Prawdopodobieństwo wystąpienia wyjątku VALUE_ERROR
  SELECT in_tekst INTO uv_tekst FROM dual;

  -- Powodujemy pojawienie się przez nas zdefiniowanego wyjątku.
  SELECT in_licz_1 INTO uv_licz FROM dual;
  IF uv_licz > 10 THEN RAISE ex_nasz_wyjatek; END IF;

  OPEN c_emp;
  LOOP
    FETCH c_emp INTO uv_emp;
    EXIT WHEN c_emp%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(c_emp%ROWCOUNT||'. '||uv_emp.last_name||'
'||uv_emp.salary);
  END LOOP;
  CLOSE c_emp;

EXCEPTION
  -- Sekcja wyjątku zawiera kod opisujący odpowiedź na wystąpienie w sekcji
  -- wykonawczej błędów. Pominięcie tej sekcji sprawia, że w przypadku
  -- wystąpienia błędu działanie programu jest przedwcześnie zakończone
  -- oraz następuje wyświetlenie informacji o błędzie.

  -- Nasz wyjątek.
  WHEN ex_nasz_wyjatek THEN
    DBMS_OUTPUT.PUT_LINE('Wartość parametru ''in_licz_1'' jest większa od 10.');

  -- Wyjatki predefiniowane.
```

```
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Zapytanie nie zwróciło danych.');
  WHEN TOO_MANY_ROWS THEN
    SELECT COUNT(*) INTO uv_licz FROM emp WHERE dept_id = in_dept_id;
    DBMS_OUTPUT.PUT_LINE('Zapytanie zwróciło '||uv_licz||' rekordów.');
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Dzielenie przez zero!');

  -- Uwaga. Gdyby OTHERS umiecić nie na końcu, to otrzymamy błąd kompilacji:
  -- PLS-00370: procedura OTHERS musi być ostatnią w ciągu definiowanych
  -- wyjątków w bloku.
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
```

Przykład 21

Testowanie powyższej procedury. Zła liczba argumentów.

```
SQL> EXECUTE testuj_wyjatki;
BEGIN testuj_wyjatki; END;

      *
BŁĄD w linii 1:
ORA-06550: linia 1, kolumna 7:
PLS-00306: zła liczba lub typy argumentów w wywołaniu 'TESTUJ_WYJATKI'
ORA-06550: linia 1, kolumna 7:
PL/SQL: Statement ignored
```

Nie pojawił się żaden błąd wykonania.

```
SQL> EXECUTE testuj_wyjatki (1500, 10, 1, 1, 'a');
1. Nguyen 1525
2. Ropeburn 1550
3. Sedeghi 1515
4. Velasquez 2500

PL/SQL procedure successfully completed.
```

Pojawia się błąd obsługiwany przez sekcję: `WHEN OTHERS THEN ...`

```
SQL> EXECUTE testuj_wyjatki(1500, 10, 1, 1, 'abc');
ORA-06502: PL/SQL: błąd  liczby lub wartości

PL/SQL procedure successfully completed.
```

Gdy usuniemy z przykładowego programu fragment:

```
WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE(SQLERRM);
```

```
SQL> EXECUTE testuj_wyjatki(1500, 10, 1, 1, 'abc');
BEGIN testuj_wyjatki(1500, 10, 1, 1, 'abc'); END;

*
BŁĄD w linii 1:
ORA-06502: PL/SQL: błąd  liczby lub wartości
ORA-06512: przy "SUMMIT2.TESTUJ_WYJATKI", linia 29
ORA-06512: przy linia 1
```

Błąd dzielenia przez zero – predefiniowany wyjątek ZERO_DIVIDE.

```
SQL> EXECUTE testuj_wyjatki(1500, 10, 1, 0, 'a');
Dzielenie przez zero!

PL/SQL procedure successfully completed.
```

Niejawny kursor zwraca więcej niż jeden rekord – predefiniowany wyjątek TOO_MANY_ROWS.

```
SQL> EXECUTE testuj_wyjatki(1500, 31, 1, 1, 'a');
Zapytanie zwróciło 2 rekordów.

PL/SQL procedure successfully completed.
```

Niejawny kursor nie zwraca żadnego rekordu – predefiniowany wyjątek NO_DATA_FOUND.

```
SQL> EXECUTE testuj_wyjatki(1500, 99, 1, 1, 'a');
Zapytanie nie zwróciło danych.

PL/SQL procedure successfully completed.
```

Wynik pojawienia się wyjątku zdefiniowanego przez użytkownika.

```
SQL> EXECUTE testuj_wyjatki(1500, 10, 100, 1, 'a');
Wartość parametru 'in_licz_1' jest większa od 10.

PL/SQL procedure successfully completed.
```

Komentarz:

Pamiętajmy, że aby zobaczyć wyniki na ekranie należy wydać polecenie: SET SERVEROUTPUT ON. Zwróćmy uwagę, że pomimo błędów wykonania otrzymujemy komunikat **PL/SQL procedure successfully completed**. Jest to spowodowane tym, że wszystkie ew. wyjątki zostają przechwycone w sekcji obsługi wyjątków EXCEPTION i nie powodują "krytycznego" przerwania działania programu. Można powiedzieć, że programy są przerywane w sposób bardziej "kontrolowany".

Gdyby z procedury usunąć obsługę WHEN OTHERS THEN, wówczas wszystkie wyjątki, które nie są jawnie obsługiwane nie będą mogły być przechwycone w procedurze i wobec tego nastąpi ich "propagacja" do środowiska programu wywołującego (np. SQL*Plus).

5.2. Funkcje SQLCODE i SQLERRM

Przykład 22

```
-- Funkcja SQLCODE zwraca numer błędu, który wystąpił. Numer jest zawsze
-- ujemny, za wyjątkiem błędu NO_DATA_FOUND (+100) i błędów definiowanych
-- przez użytkownika (+1).

-- Funkcja SQLERRM zwraca treść błędu, który wystąpił.

-- Jeżeli nie wystąpił żaden błąd to SQLCODE zwraca 0 a SQLERRM zwraca
-- "ORA-0000: normal, successful completion"

DECLARE
  err_num      NUMBER;
  err_msg      VARCHAR2(200);
  uv_last_name VARCHAR2(30);
BEGIN
```

```
  SELECT last_name INTO uv_last_name FROM emp WHERE id = &in_id;
  DBMS_OUTPUT.PUT_LINE(uv_last_name);
EXCEPTION
  WHEN OTHERS THEN
    err_num := SQLCODE;
    err_msg := SQLERRM;
    DBMS_OUTPUT.PUT_LINE('SQLCODE: '||err_num);
    DBMS_OUTPUT.PUT_LINE('SQLERRM: '||err_msg);
END;
```

Komentarz:

W przykładzie użyto tzw. zmiennej podstawianej (ang. *substitution variable*). Zmienną taką poprzedzamy znakiem &. Szczegóły można znaleźć w dokumentacji programu SQL*Plus. Poniżej pokazano przykładową sesję z programem SQL*Plus demonstrującą najważniejsze pojęcia:

```
-- Musimy pamiętać, aby zmienna DEFINE była ustawiona na ON.
-- Przepisane z dokumentacji:
-- SET DEFINE OFF disables the parsing of commands to replace substitution
-- variables with their values.

SQL>
SQL> SET DEFINE OFF
SQL> DECLARE
  2  licz NUMBER;
  3  BEGIN
  4  licz := &x;
  5  dbms_output.put_line(licz);
  6  END;
  7  /
SP2-0552: Bind variable "X" not declared.
```

```
-- Teraz wartość zmiennej DEFINE pozwala przyjmować zmienne.

SQL> SET DEFINE ON
SQL> DECLARE
  2  licz NUMBER;
  3  BEGIN
  4  licz := &x;
  5  dbms_output.put_line(licz);
  6  END;
  7  /
Enter value for x: 1234
old   4: licz := &x;
new   4: licz := 1234;
1234

PL/SQL procedure successfully completed.
```

```
-- Ustawiając zmienną VERIFY na wartość OFF "pozbywamy" się komunikatów
-- diagnostycznych. Przepisane z dokumentacji:
-- SET VER[IFY] {ON | OFF}
-- Controls whether to list the text of a SQL statement or PL/SQL command
-- before and after replacing substitution variables with values.

SQL> SET VERIFY OFF
SQL>
SQL> DECLARE
  2  licz NUMBER;
  3  BEGIN
  4  licz := &x;
```

```
  5  dbms_output.put_line(licz);
  6  END;
  7  /
Enter value for x: 5678
5678

PL/SQL procedure successfully completed.
```

Przykład 23

Przykład, który generuje błąd wykonania.

```
SQL> DECLARE
  2    err_num      NUMBER;
  3    err_msg      VARCHAR2(200);
  4    uv_last_name VARCHAR2(30);
  5  BEGIN
  6    SELECT last_name INTO uv_last_name FROM emp WHERE id = &in_id;
  7    DBMS_OUTPUT.PUT_LINE(uv_last_name);
  8  EXCEPTION
  9    WHEN OTHERS THEN
 10      err_num := SQLCODE;
 11      err_msg := SQLERRM;
 12      DBMS_OUTPUT.PUT_LINE('SQLCODE: '||err_num);
 13      DBMS_OUTPUT.PUT_LINE('SQLERRM: '||err_msg);
 14  END;
 15  /
Enter value for in_id: 'a'
old   6:   SELECT last_name INTO uv_last_name FROM emp WHERE id = &in_id;
new   6:   SELECT last_name INTO uv_last_name FROM emp WHERE id = 'a';
SQLCODE: -1722
SQLERRM: ORA-01722: invalid number

PL/SQL procedure successfully completed.
```

Przykład 24

Przykład, gdy nie wystąpił żaden błąd.

```
SQL> DECLARE
  2    uv_sysdate date;
  3  BEGIN
  4    SELECT sysdate INTO uv_sysdate FROM dual;
  5    DBMS_OUTPUT.PUT_LINE('SQLCODE: '||SQLCODE);
  6    DBMS_OUTPUT.PUT_LINE('SQLERRM: '||SQLERRM);
  7  END;
  8  /
SQLCODE: 0
SQLERRM: ORA-0000: normal, successful completion

PL/SQL procedure successfully completed.
```

5.3. Procedura RAISE_APPLICATION_ERROR

Składnia

```
RAISE_APPLICATION_ERROR(error_number, message, [, TRUE | FALSE])
```

Przykład 25

```
-- Procedura pozwala na przesyłanie komunikatów o błedach zdefiniowanych
```

```
-- przez użytkownika do programu nadrzędnego.

-- Numery błędów mogą zawierać sie w przedziale -20000 : -20999.

-- Tekst błędu może mieć do 2048 bajtów.

-- TRUE; FALSE mówi o tym, czy błąd ma być umieszczony na szczycie
-- stosu błędów, czy też ma je zastąpić.

BEGIN
  FOR uv_emp IN (SELECT * FROM emp ORDER BY salary DESC) LOOP
    IF (uv_emp.salary < 1500) THEN
      RAISE_APPLICATION_ERROR(-20555,'Pracownicy zarabiają za mało!');
    ELSE
      DBMS_OUTPUT.PUT_LINE('ID: '||uv_emp.id||', zarobek: '||uv_emp.salary);
    END IF;
  END LOOP;
END;
```

Komentarz:

Wynik poprzedniego przykładu. Wyświetliły się dane tylko czterech pracowników. Nie jest to błąd ponieważ kolejny pracownik zarabia mniej niż 1500, więc zadziałała procedura `RAISE_APPLICATION_ERROR` i program zwrócił zdefiniowany do programu wywołującego błąd o numerze `-20555`.

```
ID: 1, zarobek: 2500
ID: 5, zarobek: 1550
ID: 14, zarobek: 1525
ID: 13, zarobek: 1515
BEGIN
*
BŁĄD w linii 1:
ORA-20555: Pracownicy zarabiają za mało!
ORA-06512: przy linia 4
```

5.4. Zestawienie predefiniowanych wyjątków systemowych

Lista wyjątków predefiniowanych

Exception	Oracle	Error SQLCODE Value
ACCESS_INTO_NULL	ORA-06530	-6530
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500

SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Opis (przepisane z dokumentacji: *PL/SQL User's Guide and Reference Release 8.1.6 Part Number A77069-01*)

```
ACCESS_INTO_NULL
Your program attempts to assign values to the attributes of an uninitialized
(atomically null) object.

COLLECTION_IS_NULL
Your program attempts to apply collection methods other than EXISTS to an
uninitialized (atomically null) nested table or varray, or the program attempts
to assign values to the elements of an uninitialized nested table or varray.

CURSOR_ALREADY_OPEN
Your program attempts to open an already open cursor. A cursor must be closed
before it can be reopened. A cursor FOR loop automatically opens the cursor to
which it refers. So, your program cannot open that cursor inside the loop.

DUP_VAL_ON_INDEX
Your program attempts to store duplicate values in a database column that is
constrained by a unique index.

INVALID_CURSOR
Your program attempts an illegal cursor operation such as closing an unopened
cursor.

INVALID_NUMBER
In a SQL statement, the conversion of a character string into a number fails
because the string does not represent a valid number. (In procedural statements,
VALUE_ERROR is raised.) This exception is also raised when the LIMIT-clause
expression in a bulk FETCH statement does not evaluate to a positive number.

LOGIN_DENIED
Your program attempts to log on to Oracle with an invalid username and/or
password.

NO_DATA_FOUND
A SELECT INTO statement returns no rows, or your program references a deleted
element in a nested table or an uninitialized element in an index-by table. SQL
aggregate functions such as AVG and SUM always return a value or a null. So, a
SELECT INTO statement that calls an aggregate function never raises
NO_DATA_FOUND. The FETCH statement is expected to return no rows eventually, so
when that happens, no exception is raised.

NOT_LOGGED_ON
Your program issues a database call without being connected to Oracle.

PROGRAM_ERROR
PL/SQL has an internal problem.

ROWTYPE_MISMATCH
```

```
The host cursor variable and PL/SQL cursor variable involved in an assignment
have incompatible return types. For example, when an open host cursor variable
is passed to a stored subprogram, the return types of the actual and formal
parameters must be compatible.

SELF_IS_NULL
Your program attempts to call a MEMBER method on a null instance. That is, the
built-in parameter SELF (which is always the first parameter passed to a MEMBER
method) is null.

STORAGE_ERROR
PL/SQL runs out of memory or memory has been corrupted.

SUBSCRIPT_BEYOND_COUNT
Your program references a nested table or varray element using an index number
larger than the number of elements in the collection.

SUBSCRIPT_OUTSIDE_LIMIT
Your program references a nested table or varray element using an index number
(-1 for example) that is outside the legal range.

SYS_INVALID_ROWID
The conversion of a character string into a universal rowid fails because the
character string does not represent a valid rowid.

TIMEOUT_ON_RESOURCE
A time-out occurs while Oracle is waiting for a resource.

TOO_MANY_ROWS
A SELECT INTO statement returns more than one row.

VALUE_ERROR
An arithmetic, conversion, truncation, or size-constraint error occurs. For
example, when your program selects a column value into a character variable, if
the value is longer than the declared length of the variable, PL/SQL aborts the
assignment and raises VALUE_ERROR. In procedural statements, VALUE_ERROR is
raised if the conversion of a character string into a number fails. (In SQL
statements, INVALID_NUMBER is raised.)

ZERO_DIVIDE
Your program attempts to divide a number by zero.
```

6. Procedury i funkcje

6.1. Składnia

Składnia procedury

```
CREATE [OR REPLACE] PROCEDURE nazwa_procedury
[(param1 IN | OUT | IN OUT typ_danych [:= | DEFAULT wyrażenie],
  param2 IN | OUT | IN OUT typ_danych [:= | DEFAULT wyrażenie], ...)]
AS | IS
[zmienna1 TYP_DANYCH;
 zmienna2 TYP_DANYCH;
 ...]
BEGIN
  instrukcje;
  ...
[EXCEPTION
  WHEN nazwa_wyjątku THEN instrukcje;
  ...]
```

```
END [nazwa_procedury];

-- IN:     Wartość przekazywana do programu. Wewnątrz procedury nie można
           zmieniać parametru. Zachowuje się jak stała.

-- OUT:    Wartość zwracana do środowiska. Podanie jakiejkolwiek wartości
           tego parametru podczas wywołania procedury jest ignorowane.
           Zachowuje się jak nie inicjowana zmienna.

-- IN OUT: Wartość przekazywana do programu i zwracana do środowiska.
           Używamy, gdy zmienna ma być zarówno odczytywana jak i zmieniana,
           a następnie zwracana do programu wywołującego.
           Zachowuje się jak zainicjowana zmienna.

-- TYP_DANYCH:
          Podajemy tylko rodzaj typu, bez dokładnej specyfikacji.
          Czyli np. VARCHAR2 zamiast VARCHAR2(30), NUMBER zamiast NUMEBR(5).
```

Składnia funkcji

```
CREATE [OR REPLACE] FUNCTION nazwa_funkcji
[(param1 IN | OUT | IN OUT typ_danych [:= | DEFAULT wyrażenie],
  param2 IN | OUT | IN OUT typ_danych [:= | DEFAULT wyrażenie], ...)]
RETURN typ_danych AS | IS
BEGIN
  instrukcje;
  ...
  RETURN wyrazenie;
[EXCEPTION
  WHEN nazwa_wyjątku THEN instrukcje;
  ...]
END [nazwa_funkcji];
```

6.2. Najprostsza procedura

Przykład 26

Przykład procedury „nic-nie-robiącej”

```
CREATE OR REPLACE PROCEDURE nic_nie_robi IS
BEGIN
  NULL;  -- Musi być. Nie można nic nie wpisać.
END;
```

Załadowanie oraz sprawdzenie statusu:

```
SQL> CREATE OR REPLACE PROCEDURE nic_nie_robi IS
  2  BEGIN
  3    NULL;
  4  END;
  5  /

Procedure created.

SQL> SELECT object_name, object_type, status
     FROM user_objects WHERE object_name = 'NIC_NIE_ROBI';

OBJECT_NAME                                    OBJECT_TYPE        STATUS
---------------------------------------------- ------------------ -------
NIC_NIE_ROBI                                   PROCEDURE          VALID
```

Przykład, gdy załadowana procedura zawiera błędy:

```
SQL> CREATE OR REPLACE PROCEDURE nic_nie_robi IS
  2  BEGIN
  3    NULLL;
  4  END;
  5  /

Warning: Procedure created with compilation errors.

SQL> SELECT object_name, object_type, status
     FROM user_objects WHERE object_name = 'NIC_NIE_ROBI';

OBJECT_NAME                      OBJECT_TYPE        STATUS
-------------------------------- ------------------ -------
NIC_NIE_ROBI                     PROCEDURE          INVALID

SQL> SHOW ERRORS;
Errors for PROCEDURE NIC_NIE_ROBI:

LINE/COL ERROR
-------- -----------------------------------------------------------------
3/3      PLS-00201: identifier 'NULLL' must be declared
3/3      PL/SQL: Statement ignored
SQL>
```

6.3. Parametry procedur

Przykład 27

```
CREATE OR REPLACE PROCEDURE callme (a NUMBER, b NUMBER) IS
BEGIN
  NULL;
END;

Procedure created.

SQL> EXECUTE callme();
BEGIN callme(); END;

PLS-00306: wrong number or types of arguments in call to 'CALLME'
```

```
-- Notacja pozycyjna (ang. Positional Notation)
SQL> EXECUTE callme (2, 5);

PL/SQL procedure successfully completed.

-- Notacja imienna (ang. Named Notation)
SQL> EXECUTE callme (a => 2, b => 5);

PL/SQL procedure successfully completed.

-- Notacja imienna
SQL> EXECUTE callme (b => 2, a => 5);

PL/SQL procedure successfully completed.
```

```
-- Notacja mieszana (ang. Mixed Notation)
SQL> EXECUTE callme (2, b => 5);

PL/SQL procedure successfully completed.

-- Notacja mieszana
```

```
SQL> EXECUTE callme (b => 5, 2);
BEGIN callme(b => 5, 2); END;

PLS-00312: a positional parameter association may not follow a named association

-- Notacja mieszana
SQL> EXECUTE callme (2, a => 5);
BEGIN callme(2, a => 5); END;

PLS-00703: multiple instances of named argument in list
```

6.4. Typy zmiennych IN, OUT oraz IN OUT

Przykład 28
Jak odwoływać się do zmiennych typu **IN**

```
--------------------------------------------------------------------------
-- Tworzymy procedurę. Niestety nie udaje się to. Zmiennej typu IN nie można
-- modyfikować wewnątrz procedury.
--------------------------------------------------------------------------
SQL> CREATE OR REPLACE PROCEDURE callme (test IN NUMBER) IS
  2  BEGIN
  3    test := 4;
  4  END;
  5  /

Warning: Procedure created with compilation errors.

SQL> SHOW ERRORS
Errors for PROCEDURE CALLME:

LINE/COL ERROR
-------- -----------------------------------------------------------------
3/3      PLS-00363: expression 'TEST' cannot be used as an assignment target
3/3      PL/SQL: Statement ignored
SQL>
```

```
--------------------------------------------------------------------------
-- Tworzymy procedurę. Tym razem jest OK.
--------------------------------------------------------------------------
CREATE OR REPLACE PROCEDURE callme (test IN NUMBER) IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('Wartosc zmiennej test: ' || test);
END;

Procedure created.

--------------------------------------------------------------------------
-- Wywołujemy procedurę.
--------------------------------------------------------------------------
SQL> EXECUTE callme(10);
Wartosc zmiennej test: 10

PL/SQL procedure successfully completed.
```

Przykład 29
Jak odwoływać się do zmiennych typu **OUT**

```
--------------------------------------------------------------------------
-- Tworzymy procedurę
```

```
--------------------------------------------------------------------------------
CREATE OR REPLACE PROCEDURE callme (test OUT NUMBER) IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('Wartosc zmiennej test przed: ' || test);
  -- Przetestuj, gdy będzie: test := test + 10. Jaki stąd wniosek ?
  test := 10;
  DBMS_OUTPUT.PUT_LINE('Wartosc zmiennej test po: ' || test);
END;

Procedure created.


--------------------------------------------------------------------------------
-- Wywołujemy procedurę, która zwraca do środowiska wywołującego
-- wartość zmiennej. Okazuje się, że podanie jakiejkolwiek wartości
-- tego parametru podczas wywołania procedury jest ignorowane
--------------------------------------------------------------------------------
DECLARE
x NUMBER := 5;
BEGIN
  callme(x);
  DBMS_OUTPUT.PUT_LINE('Wartosc zmiennej x: ' || x);
END;

Wartosc zmiennej test przed:
Wartosc zmiennej test po: 10
Wartosc zmiennej x: 10


PL/SQL procedure successfully completed.
```

```
--------------------------------------------------------------------------------
-- Ale tak jak poniżej jest niedozwolone. Parametr OUT musi być zmienną.
-- Nie może to być stała ani też wyrażenie (np. zm1 + zm2).
--------------------------------------------------------------------------------
SQL> EXECUTE callme (2);
BEGIN callme (2); END;

PLS-00363: expression '2' cannot be used as an assignment target
```

Przykład 30
Jak odwoływać się do zmiennych typu **IN OUT**

```
--------------------------------------------------------------------------------
-- Tworzymy procedurę
--------------------------------------------------------------------------------
CREATE OR REPLACE PROCEDURE callme (test IN OUT NUMBER) IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('Wartosc zmiennej test przed: ' || test);
  test := test + 10;
  DBMS_OUTPUT.PUT_LINE('Wartosc zmiennej test po: ' || test);
END;

Procedure created.
```

```
--------------------------------------------------------------------------------
-- Wywołujemy procedurę, która zwraca do środowiska wywołującego
-- wartość zmiennej. Ponieważ zmienna jest typu IN OUT, więc
-- wartości tego parametru podczas wywołania procedury NIE jest ignorowana.
```

```
--------------------------------------------------------------------------------
DECLARE
x NUMBER := 5;
BEGIN
  callme(x);
  DBMS_OUTPUT.PUT_LINE('Wartosc zmiennej x: ' || x);
END;
/

Wartosc zmiennej test przed: 5
Wartosc zmiennej test po: 15
Wartosc zmiennej x: 15

PL/SQL procedure successfully completed.

--------------------------------------------------------------------------------
-- Tutaj też nie jest dozwolone jak poniżej.
--------------------------------------------------------------------------------
SQL> EXECUTE callme (2);
BEGIN callme (2); END;

PLS-00363: expression '2' cannot be used as an assignment target
```

6.5. Inne przykłady procedur

Przykład 31

Przykład procedury – pracownicy na danym stanowisku i suma ich zarobków

```
CREATE OR REPLACE PROCEDURE pracownicy_na_stanowiskach
(in_title IN VARCHAR2) IS
-- Procedura wyświetla dane pracowników pracujących na stanowisku
-- podanym jako parametr wejściowy. Dodatkowo wyświetlana jest suma
-- zarobków tych pracowników.

CURSOR c_emp IS
  SELECT first_name, last_name, salary, title
  FROM emp
  WHERE UPPER(title) = UPPER(in_title);  -- odczulamy na wielkość liter

uv_emp c_emp%ROWTYPE;
uv_title emp.title%TYPE;
uv_salary emp.salary%TYPE;

BEGIN
  -- Pobranie nazwy stanowiska w takim brzmieniu, jak w bazie.
  -- GROUP BY, aby na pewno SELECT zwrócił dokładnie 1 rekord.
  SELECT title INTO uv_title
  FROM emp
  WHERE UPPER(title) = UPPER(in_title) GROUP BY title;

  DBMS_OUTPUT.PUT_LINE('Pracownicy na stanowisku: '||uv_title);
  DBMS_OUTPUT.PUT_LINE('-----------------------------------------');

  OPEN c_emp;
  LOOP
    FETCH c_emp INTO uv_emp;
      EXIT WHEN c_emp%NOTFOUND;
      -- Długość wynikowego tekstu będzie 25+7=32 znaki.
      -- LPAD, aby liczby były wyrównane do lewej.
      DBMS_OUTPUT.PUT_LINE(
        RPAD(uv_emp.first_name||' '||uv_emp.last_name,25)||
```

```
        LPAD(uv_emp.salary,7));
  END LOOP;
  CLOSE c_emp;

  -- Wyliczamy sumę zarobków dla podanego stanowiska.
  -- Jeszcze raz wykorzystujemy zmienną uv_emp.title.
  SELECT SUM(salary) INTO uv_salary FROM emp WHERE title = uv_emp.title;

  -- Rysujemy linię o długości 32 znaki. Sumę zarobków równamy do lewej.
  DBMS_OUTPUT.PUT_LINE(RPAD('-',32,'-'));
  DBMS_OUTPUT.PUT_LINE(RPAD('Suma zarobków: ',25)||LPAD(uv_salary,7));
END;
```

Przykład 32

Wywołanie procedury

```
EXECUTE pracownicy_na_stanowiskach('StoCk CleRK');
```

lub

```
BEGIN pracownicy_na_stanowiskach('StoCk CleRK'); END;
/
```

```
BEGIN pracownicy_na_stanowiskach('aqq'); END;
/

Gdy podamy nieistniejące stanowisko, to otrzymamy błąd:
BŁĄD w linii 1:
ORA-01403: nie znaleziono żadnych danych
ORA-06512: przy "SUMMIT2.PRACOWNICY_NA_STANOWISKACH", linia 19
ORA-06512: przy linia 1
```

Przykład 33

Przykład procedury – Odnajduje *n* (*n* przekazane jako parametr) najlepiej sprzedających się produktów w poszczególnych hurtowniach i wpisuje je do tabeli tymczasowej (musi ona być wcześniej stworzona).

```
CREATE OR REPLACE PROCEDURE Top_n (in_n IN NUMBER) IS
-- Odnajduje n (n przekazane jako parametr) najlepiej sprzedających się
-- produktów w poszczególnych hurtowniach i wpisuje je do tabeli tymczasowej
-- 'temp_top_n'. Musi ona być wcześniej stworzona. W PL/SQL NIE MOŻNA używać
-- poleceń DDL!

-- CREATE TABLE temp_top_n (
--   hurtownia VARCHAR2(30),
--   nazwa_produktu VARCHAR2(50),
--   sprzedaz NUMBER(9));

CURSOR c_warehouse IS
  SELECT *
  FROM warehouse
  ORDER BY city;

CURSOR c_inventory (in_warehouse_id NUMBER) IS
  SELECT P.name, max_in_stock - amount_in_stock "sale"
  FROM inventory I, product P
  WHERE warehouse_id = in_warehouse_id AND
        P.id = I.product_id
  ORDER BY max_in_stock - amount_in_stock DESC;
```

```
uv_inventory c_inventory%ROWTYPE;
uv_warehouse c_warehouse%ROWTYPE;

BEGIN
  OPEN c_warehouse;
  DELETE FROM temp_top_n;
  -- Pętla zewnętrzna - "przechodzi" po hurtowniach.
  LOOP
    FETCH c_warehouse INTO uv_warehouse;
    EXIT WHEN c_warehouse%NOTFOUND; -- EXIT musi być po FETCH.
    OPEN c_inventory(uv_warehouse.id);
    -- Pętla wewnętrzna. "Przechodzi" po stanach magazynowych.
    -- Musimy pobrać tylko in_n rekordów. Patrz warunek ORDER BY
    -- w definicji kursora c_inventory.
    FOR i IN 1..in_n LOOP
      FETCH c_inventory INTO uv_inventory;
      EXIT WHEN c_inventory%NOTFOUND; -- EXIT musi być po FETCH.
      INSERT INTO temp_top_n VALUES
         (uv_warehouse.city, uv_inventory.name, uv_inventory."sale");
    END LOOP;
  CLOSE c_inventory;
  END LOOP;
  CLOSE c_warehouse;
  COMMIT;
END;
```

6.6. Przykłady funkcji

Przykład 34

Przykład funkcji – minimalna praca w zespole

```
CREATE OR REPLACE FUNCTION min_placa_w_zespole (in_dept_id IN  NUMBER)
RETURN NUMBER IS
-- Funkcja zwraca minimalną płacę jaka jest w zespole (kolumna DEPT_ID)
-- przekazanym jako parametr funkcji.

uv_min_placa emp.salary%TYPE;
BEGIN
  SELECT MIN(salary) INTO uv_min_placa
  FROM emp
  WHERE dept_id = in_dept_id;
  RETURN (uv_min_placa);
END;
```

Komentarz:

Wywołanie funkcji wykonujemy w następujący sposób:

```
SELECT min_placa_w_zespole(31) "Płaca minimalna" FROM dual;

Płaca minimalna
---------------
           1400
```

7. Wykorzystanie pakietów w celu organizacji kodu

7.1. Składnia

Składnia pakietu

```
-- Specyfikacja pakietu.
CREATE [OR REPLACE] PACKAGE nazwa_pakietu AS | IS
  1. deklaracje stałych, zmiennych, kursorów
     dostępnych na zewnątrz pakietu
  2. deklaracje publicznych procedur i funkcji
END [nazwa_pakietu];

-- Ciało pakietu.
CREATE [OR REPLACE] PACKAGE BODY nazwa_pakietu AS | IS
  1. deklaracje stałych, zmiennych, kursorów
     dostępnych tylko wewnątrz pakietu
  2. definicje publicznych procedur i funkcji
  3. definicje prywatnych procedur i funkcji
END [nazwa_pakietu];
```

Komentarz:

- Pakiet grupuje powiązane logicznie procedury, funkcje i kursory.
- Ukrywa szczegóły implementacji. Użytkownikowi są udostępniane tylko specyfikacje pakietów.
- Zwiększają funkcjonalność. Zmienne zadeklarowane w pakietach istnieją przez całą sesję użytkownika. Dzięki temu mogą one służyć do wzajemnej komunikacji różnych procesów.
- Większa szybkość działania.
- Procedury i funkcje mogą być umieszczane w dowolnej kolejności w specyfikacji oraz w ciele pakietu. Ciało pakietu musi zawierać definicje wszystkich programów wymienionych w specyfikacji. Odwrotna zasada nie musi już być spełniona. Ciało pakietu może zawierać programy, które nie są wymienione w specyfikacji. Takie programy mają charakter programów prywatnych i są dostępne TYLKO wewnątrz pakietu, gdzie są zdefiniowane. Innymi słowy programu prywatnego nie można w żaden sposób wywołać "ręcznie" (np. z poziomu innego pakietu lub z poziomu programu SQL*Plus).

7.2. Przykłady

Przykład 35

```
CREATE OR REPLACE PACKAGE testowy AS

data_start DATE := TO_DATE(01-01-2005,'DD-MM-YYYY');

-- Przeciążanie procedur. Obie procedury maja tą samą nazwę ale różne parametry.
PROCEDURE dodaj (id NUMBER, imie VARCHAR2, nazwisko VARCHAR2);
PROCEDURE dodaj (id NUMBER, data DATE);
FUNCTION zarobki (dzial_id IN NUMBER) RETURN NUMBER;

END testowy;
```

```
CREATE OR REPLACE PACCKAGE BODY testowy AS

-- Definicja pierwszej procedury
PROCEDURE dodaj (id NUMBER, imie VARCHAR2, nazwisko VARCHAR2) IS
...
END;
```

```
-- Definicja drugiej procedury
PROCEDURE dodaj (id NUMBER, data DATE) IS
...
END;

-- Definicja funkcji
FUNCTION zarobki (dzial_id NUMBER) IS
...
  RETURN wyliczone_zarobki;
END;

-- Procedura prywatna. Nie jest ona zdefiniowana w specyfikacji pakietu.
PROCEDURE tajne;
...
END;
```

Przykład 36

Tworzymy procedurę do wstawiania danych do tabeli EMP oraz funkcję, która wyznacza minimalną płacę w podanym jako parametr dziale. Specyfikacja pakietu.

```
CREATE OR REPLACE PACKAGE nasz_pakiet AS

-- Procedura tworzy nowego pracownika. Tylko pola NOT NULL są jawnie wymagane.
PROCEDURE nowy_pracownik (
  in_last_name      IN VARCHAR2,
  in_first_name     IN VARCHAR2,
  in_userid         IN VARCHAR2 DEFAULT NULL,
  in_start_date     IN DATE     DEFAULT NULL,
  in_comments       IN VARCHAR2 DEFAULT NULL,
  in_manager_id     IN NUMBER   DEFAULT NULL,
  in_title          IN VARCHAR2 DEFAULT NULL,
  in_dept_id        IN NUMBER   DEFAULT NULL,
  in_salary         IN NUMBER   DEFAULT NULL,
  in_commission_pct IN NUMBER   DEFAULT NULL);

-- Funkcja zwraca minimalną płacę jaka jest w zespole (kolumna DEPT_ID)
-- przekazanym jako parametr funkcji.
FUNCTION min_placa_w_zespole (in_dept_id IN  NUMBER) RETURN NUMBER;

-- Zmienna, która będzie dostępna na zewnątrz pakietu.
ex_bad_user EXCEPTION;
END nasz_pakiet;
```

Ciało pakietu.

```
CREATE OR REPLACE PACKAGE BODY nasz_pakiet AS

PROCEDURE nowy_pracownik (
  in_last_name      IN VARCHAR2,
  in_first_name     IN VARCHAR2,
  in_userid         IN VARCHAR2 DEFAULT NULL,
  in_start_date     IN DATE     DEFAULT NULL,
  in_comments       IN VARCHAR2 DEFAULT NULL,
  in_manager_id     IN NUMBER   DEFAULT NULL,
  in_title          IN VARCHAR2 DEFAULT NULL,
  in_dept_id        IN NUMBER   DEFAULT NULL,
  in_salary         IN NUMBER   DEFAULT NULL,
  in_commission_pct IN NUMBER   DEFAULT NULL) AS

uv_max_id NUMBER;
```

```
BEGIN
  SELECT MAX(id) INTO uv_max_id FROM emp;
  INSERT INTO emp
  VALUES (uv_max_id + 1, in_last_name, in_first_name,
          in_userid, in_start_date, in_comments, in_manager_id,
          in_title, in_dept_id, in_salary, in_commission_pct);
  COMMIT;
END nowy_pracownik;

FUNCTION min_placa_w_zespole (
  in_dept_id IN  NUMBER) RETURN NUMBER AS
uv_min_placa emp.salary%TYPE;
BEGIN
  SELECT MIN(salary) INTO uv_min_placa
  FROM emp
  WHERE dept_id = in_dept_id;
  RETURN (uv_min_placa);
END min_placa_w_zespole;

END nasz_pakiet;
```

Wywoływanie procedur i funkcji ze zdefiniowanego pakietu – przykłady

```
SELECT nasz_pakiet.min_placa_w_zespole(31) FROM DUAL;

BEGIN
  nasz_pakiet.nowy_pracownik(
    'Gramacki', 'Artur', 'orauser', TO_DATE('01-01-1999','DD-MM-YYYY'));
END;
```

```
BEGIN
  nasz_pakiet.nowy_pracownik(
    in_start_date => TO_DATE('01-01-1999','DD-MM-YYYY'), -- parametr nr 4
    in_first_name => 'Artur',                            -- parametr nr 2
    in_last_name  => 'Gramacki',                         -- parametr nr 1
    in_salary     => 10000);                             -- parametr nr 9
END;
```

Komentarz:

W pierwszym przykładzie użyto tzw. **notacja pozycyjna** (ang. *Positional Notation*). Argumenty musza być podawane w ściśle określonej kolejności.

W drugim przykładzie użyto tzw. **notacja imienna** (ang. *Named Notation*). Zaleta tej metody jest to, że poszczególne argumenty można podawać w dowolnej kolejności. Warunek - musimy znać nazwy używanych parametrów procedury lub funkcji. Ta notacja jest przydatna, gdy procedura lub funkcja ma dużo argumentów a my chcemy podąć tylko kilka z nich.

7.3. Kompilowanie i usuwanie procedur, funkcji i pakietów

Składnia

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

Komentarz:

System Oracle dokonuje kompilacji kodów PL/SQL w momencie ładowania go do serwera (polecenie `CREATE OR REPLACE ...`) . Nie trzeba więc tego robić ręcznie.

Przykład 37

```
c:\Database\Ora8i\bin>wrap.exe iname=nasz_pakiet.sql

PL/SQL Wrapper: Release 8.1.7.0.0 - Production on Ndz Kwi 17 08:49:19 2005

Copyright (c) Oracle Corporation 1993, 2000.  All Rights Reserved.

Processing nasz_pakiet.sql to nasz_pakiet.plb

c:\Database\Ora8i\bin>
```

Oto jak wygląda zakodowane ciało pakietu (pokazano tylko fragment):

```
CREATE OR REPLACE PACKAGE BODY nasz_pakiet wrapped
0
abcd
abcd
...
abcd
abcd
3
b
8106000
1
4
0
21
2 :e:
1PACKAGE:
1BODY:
1NASZ_PAKIET:
1COMMIT:
1FUNCTION:
1MIN_PLACA_W_ZESPOLE:
1RETURN:
1MIN:
1DEPT_ID:
1=:
0

0
0
aa
2
0 a0 1d a0 97 9a 8f a0
b0 3d 8f a0 b0 3d 8f a0
1 72 1 78
1 85 1 88
...
/
```

Komentarz:

Dobrą zasadą jest pozostawienie specyfikacji pakiety w wersji jawnej. Dzięki temu użytkownik może dość łatwo zorientować się co dany pakiet robi. Mamy więc rodzaj dokumentacji pakietu.

7.4. Gdzie ORACLE przechowuje źródła PL/SQL

```
DESC user_source

Name                                      Null?    Type
----------------------------------------- -------- ------------------------
NAME                                               VARCHAR2(30)
```

TYPE	VARCHAR2(12)
LINE	NUMBER
TEXT	**VARCHAR2(4000)**

```
SELECT * FROM user_source ORDER BY name, type, line;
```

NAME	TYPE	LINE	TEXT
NASZ_PAKIET	PACKAGE	1	PACKAGE nasz_pakiet AS
NASZ_PAKIET	PACKAGE	2	-- Tworzy nowego pracwnika. Tylko pola NOT NULL sa jawnie wymagane.
NASZ_PAKIET	PACKAGE	3	PROCEDURE nowy_pracownik (
NASZ_PAKIET	PACKAGE	4	in_last_name IN VARCHAR2,
NASZ_PAKIET	PACKAGE	5	in_first_name IN VARCHAR2,
NASZ_PAKIET	PACKAGE	6	in_userid IN VARCHAR2 DEFAULT NULL,
NASZ_PAKIET	PACKAGE	7	in_start_date IN DATE DEFAULT NULL,
NASZ_PAKIET	PACKAGE	8	in_comments IN VARCHAR2 DEFAULT NULL,
NASZ_PAKIET	PACKAGE	9	in_manager_id IN NUMBER DEFAULT NULL,
NASZ_PAKIET	PACKAGE	10	in_title IN VARCHAR2 DEFAULT NULL,
NASZ_PAKIET	PACKAGE	11	in_dept_id IN NUMBER DEFAULT NULL,
NASZ_PAKIET	PACKAGE	12	in_salary IN NUMBER DEFAULT NULL,
NASZ_PAKIET	PACKAGE	13	in_commission_pct IN NUMBER DEFAULT NULL);
NASZ_PAKIET	PACKAGE	14	
NASZ_PAKIET	PACKAGE	15	-- Funkcja zwraca minimalną płacę jaka jest w zespole (kolumna DEPT_ID)
NASZ_PAKIET	PACKAGE	16	-- przekazanym jako parametr funkcji.
NASZ_PAKIET	PACKAGE	17	FUNCTION min_placa_w_zespole (in_dept_id IN NUMBER) RETURN NUMBER;
NASZ_PAKIET	PACKAGE	18	
NASZ_PAKIET	PACKAGE	19	-- Zmienna, ktora bedzie dostepna na zewnątrz pakietu.
NASZ_PAKIET	PACKAGE	20	ex_bad_user EXCEPTION;
NASZ_PAKIET	PACKAGE	21	END nasz_pakiet;
NASZ_PAKIET	PACKAGE BODY	1	PACKAGE BODY nasz_pakiet AS
NASZ_PAKIET	PACKAGE BODY	2	PROCEDURE nowy_pracownik (
NASZ_PAKIET	PACKAGE BODY	3	in_last_name IN VARCHAR2,
NASZ_PAKIET	PACKAGE BODY	4	in_first_name IN VARCHAR2,
NASZ_PAKIET	PACKAGE BODY	5	in_userid IN VARCHAR2 DEFAULT NULL,
NASZ_PAKIET	PACKAGE BODY	6	in_start_date IN DATE DEFAULT NULL,
NASZ_PAKIET	PACKAGE BODY	7	in_comments IN VARCHAR2 DEFAULT NULL,
NASZ_PAKIET	PACKAGE BODY	8	in_manager_id IN NUMBER DEFAULT NULL,
NASZ_PAKIET	PACKAGE BODY	9	in_title IN VARCHAR2 DEFAULT NULL,
NASZ_PAKIET	PACKAGE BODY	10	in_dept_id IN NUMBER DEFAULT NULL,
NASZ_PAKIET	PACKAGE BODY	11	in_salary IN NUMBER DEFAULT NULL,
NASZ_PAKIET	PACKAGE BODY	12	in_commission_pct IN NUMBER DEFAULT NULL) AS
NASZ_PAKIET	PACKAGE BODY	13	uv_max_id NUMBER;
NASZ_PAKIET	PACKAGE BODY	14	BEGIN
NASZ_PAKIET	PACKAGE BODY	15	SELECT MAX(id) INTO uv_max_id FROM emp;
NASZ_PAKIET	PACKAGE BODY	16	INSERT INTO emp
NASZ_PAKIET	PACKAGE BODY	17	VALUES (uv_max_id + 1, in_last_name, in_first_name,
NASZ_PAKIET	PACKAGE BODY	18	in_userid, in_start_date, in_comments, in_manager_id,

NASZ_PAKIET	PACKAGE BODY	19	in_title, in_dept_id, in_salary, in_commission_pct);
NASZ_PAKIET	PACKAGE BODY	20	COMMIT;
NASZ_PAKIET	PACKAGE BODY	21	END nowy_pracownik;
NASZ_PAKIET	PACKAGE BODY	22	
NASZ_PAKIET	PACKAGE BODY	23	FUNCTION min_placa_w_zespole (
NASZ_PAKIET	PACKAGE BODY	24	in_dept_id IN NUMBER) RETURN NUMBER AS
NASZ_PAKIET	PACKAGE BODY	25	uv_min_placa emp.salary%TYPE;
NASZ_PAKIET	PACKAGE BODY	26	BEGIN
NASZ_PAKIET	PACKAGE BODY	27	SELECT MIN(salary) INTO uv_min_placa
NASZ_PAKIET	PACKAGE BODY	28	FROM emp
NASZ_PAKIET	PACKAGE BODY	29	WHERE dept_id = in_dept_id;
NASZ_PAKIET	PACKAGE BODY	30	RETURN (uv_min_placa);
NASZ_PAKIET	PACKAGE BODY	31	END min_placa_w_zespole;
NASZ_PAKIET	PACKAGE BODY	32	
NASZ_PAKIET	PACKAGE BODY	33	END nasz_pakiet;

8. Kolekcje

8.1. Wstęp

(większość przykładów zaczerpnięto z [3])

Kolekcje są strukturami danych, które mogą przechowywać pewną liczbę wierszy danych w pojedynczej zmiennej. Najbliższą analogią kolekcji w języku PL/SQL są znane z innych języków programowania *tablice*. W PL/SQL istnieją trzy rodzaje kolekcji:

- tablice indeksowane (ang. *index-by tables*),
- tablice zagnieżdżone (ang. *nested tables*),
- tablice o zmiennym rozmiarze (ang. *varying arrays VARRAYs*).

Poszczególne części składowe kolekcji noszą nazwę *elementów*. Podczas korzystania z kolekcji, w celu odwołania do danego elementu należy podać w nawiasie liczbę całkowitą zwaną *indeksem*. Przykładowo czwarty element kolekcji `moja_kolekcja` zapisać należy jako `moja_kolekcja(4)`.

8.2. Przykłady

Przykład 38

Deklaracja i kasowanie własnego typu danych.

```
CREATE TYPE emp_of_dept_t AS TABLE OF VARCHAR2(100);
/

DROP TYPE emp_of_dept_t;
```

Komentarz:

Przykład 39

Funkcja zwracająca kolekcję.

```
CREATE OR REPLACE FUNCTION emp_dept (in_dept_id IN dept.id%TYPE)
```

```
RETURN emp_of_dept_t
IS
emp_of_dept emp_of_dept_t;
BEGIN
  SELECT last_name
  BULK COLLECT INTO emp_of_dept
  FROM emp E
  WHERE dept_id = in_dept_id;

  RETURN emp_of_dept;
END;
/
```

Komentarz:

Funkcja zwraca listę ciągów znakowych typu VARCHAR2, zawierających nazwiska pracowników pracujących w podanym jako parametr dziale.

Przykład 40

Blok anonimowy, który wykorzystuje utworzoną funkcję.

```
DECLARE
pracownicy emp_of_dept_t;
BEGIN
pracownicy := emp_dept(41);

FOR x IN pracownicy.FIRST .. pracownicy.LAST
LOOP
  DBMS_OUTPUT.PUT_LINE(pracownicy(x));
END LOOP;

END;
/

Ngao
Urguhart
Maduro
Smith

PL/SQL procedure successfully completed.
```

Komentarz:

Zwróćmy uwagę na tzw. metody wewnętrzne FIRST oraz LAST.

Przykład 41

Składania tworzenia kolekcji

```
-- Tablice indeksowe

DECLARE TYPE
  emp_last_name_t IS TABLE OF emp.last_name%TYPE INDEX BY BINARY_INTEGER;
  emp_last_name emp_last_name_t;
BEGIN
  null;
END;
/

-- Tablice indeksowe. Kolekcja rekordów.
-- Pozwala emulować tabelę bazy danych w programie PL/SQL.
-- Tabela taka jest przechowywana w pamięci operacyjnej i w związku z tym
```

```
-- dostęp do niej jest bardzo szybki.
DECLARE TYPE
  my_emp_t IS TABLE OF emp%ROWTYPE INDEX BY BINARY_INTEGER;
  my_emp my_emp_t;
BEGIN
  null;
END;
/
```

```
-- Współużytkowanie tablic indeksowanych.
-- Należy umieścić definicję typu danych w specyfikacji pakietu.

CREATE OR REPLACE PACKAGE my_types
AS
  TYPE my_emp_t IS TABLE OF emp%ROWTYPE INDEX BY BINARY_INTEGER;
END;
/

-- Przykład użycia tak zadeklarowanego typu.
PROCEDURE nazwa-procedury (my_emp_in my_types.my_emp_t) IS ...
FUNCTION nazwa-funkcji RETURN my_types.my_emp_t IS ...
```

```
-- Tablice zagnieżdżone

-- Deklaracja z poziomu PL/SQL (rzadko stosowane)
DECLARE TYPE
  emp_of_dept_t AS TABLE OF VARCHAR2(100);
...

-- Deklaracja jako obiekt "trwały" (stosowana częściej)
CREATE TYPE emp_of_dept_t AS TABLE OF VARCHAR2(100);
/
```

```
-- Tablice o zmiennym rozmiarze

CREATE TYPE nazwa-typu AS VARRAY(max) OF typ-danych;
```

Przykład 42

Inicjalizacja oraz przypisywanie wartości kolekcjom. Tablice indeksowane.

```
DECLARE
  TYPE emp_last_name_t IS TABLE OF emp.last_name%TYPE INDEX BY BINARY_INTEGER;
  emp_last_name emp_last_name_t;
BEGIN
  emp_last_name(1) := 'Gramacki';      -- Można używać dowolnego indeksu
  emp_last_name(33) := 'Nowak';        -- bez żadnego zmniejszenia
  emp_last_name(6) := 'Kowalski';      -- wydajności lub zwiększenia
END;                                   -- zapotrzebowania na pamięć.
/
```

Komentarz:

Można używać dowolnego indeksu bez żadnego zmniejszenia wydajności lub zwiększenia zapotrzebowania na pamięć.

Przykład 43

Inicjalizacja oraz przypisywanie wartości kolekcjom. Tablice zagnieżdżone.

```
CREATE TYPE emp_last_name_t AS TABLE OF VARCHAR2(100);
/

DECLARE
  emp_last_name emp_last_name_t;
BEGIN
  emp_last_name := emp_last_name_t();    -- tzw. konstruktor
  emp_last_name.EXTEND(3);               -- przydzielenie pamięci

  emp_last_name(1) := 'Gramacki';
  emp_last_name(2) := 'Nowak';
  emp_last_name(3) := 'Kowalski';
END;
/

-- Przykład jak wyżej. Dane podajemy jako parametry konstruktora.
-- Operacja EXTEND wykonywana jest niejawnie.
...
emp_last_name := emp_last_name_t('Gramacki', 'Nowak', 'Kowalski');
...
```

Komentarz:

Podczas przypisywanie wartości elementom tablic zagnieżdżonych lub o zmiennym rozmiarze (VARRAYs) należy zapewnić odpowiednią ilość pamięci do przechowania danych. System Oracle oferuje specjalną funkcję zwaną *konstruktorem*, która wykonuje to zadanie. Konstruktor nosi taką samą nazwę jak typ kolekcji i Oracle tworzy go *automatycznie* w momencie tworzenia typu kolekcji.

Przykład 44

Przypisywanie wartości kolekcji rekordów

```
DECLARE TYPE
  my_emp_t IS TABLE OF emp%ROWTYPE INDEX BY BINARY_INTEGER;
  my_emp my_emp_t;
BEGIN
  my_emp(1).id := 10000;
  my_emp(1).last_name := 'Gramacki';
  my_emp(1).first_name := 'Artur';
END;
```

Komentarz:

Przykład 45

Metody wewnętrzne.

```
nazwa-kolekcji.EXISTS(i)
nazwa-kolekcji.COUNT
nazwa-kolekcji.FIRST
nazwa-kolekcji.LAST
nazwa-kolekcji.PRIOR(i)
nazwa-kolekcji.NEXT(i)
nazwa-kolekcji.EXTEND(n)
nazwa-kolekcji.LIMIT

IF nazwa-kolekcji IS [NOT] NULL THEN ...
```

Pewien przykład praktyczny.

```
DECLARE
```

```
TYPE my_emp_t IS TABLE OF emp%ROWTYPE INDEX BY BINARY_INTEGER;
my_emp my_emp_t;

TYPE litery_t IS VARRAY(7) OF VARCHAR2(1);
litery litery_t := litery_t('a','b','c','d');

indeks NUMBER;

BEGIN
  my_emp(1).id := 1;
  my_emp(1).last_name := 'Gramacki';
  my_emp(1).first_name := 'Artur';

  my_emp(4).id := 4;
  my_emp(4).last_name := 'xxx';

  my_emp(10).id := 77;
  my_emp(10).last_name := 'Nowak';

  DBMS_OUTPUT.PUT_LINE(my_emp(1).id);

  IF my_emp.EXISTS(999) THEN
    DBMS_OUTPUT.PUT_LINE(my_emp(999).id);
  ELSE
    DBMS_OUTPUT.PUT_LINE('Brak 999');
  END IF;

  IF my_emp IS NOT NULL THEN
    DBMS_OUTPUT.PUT_LINE('Liczba elementów: ' || my_emp.COUNT);
    DBMS_OUTPUT.PUT_LINE('Pierwszy i ostatni element: '
      || my_emp.FIRST || ' ' ||my_emp.LAST);
  END IF;

  -- Pojawi się błąd: 'ORA-01403: nie znaleziono danych' ponieważ instrukcja
  -- pętli próbuje odczytać wszystkie elementy pomiędzy FIRST a LAST (ale u nas
  -- niektóre elementy z tego zakresu nie istnieją więc NO_DATA_FOUND).
  -- FOR indeks IN my_emp.FIRST .. my_emp.LAST
  -- LOOP
  --  DBMS_OUTPUT.PUT_LINE(my_emp(indeks).id);
  -- END LOOP;

  indeks := my_emp.FIRST;
  WHILE indeks IS NOT NULL
  LOOP
    DBMS_OUTPUT.PUT_LINE(my_emp(indeks).id);
    indeks := my_emp.NEXT(indeks);
  END LOOP;

  -- Tylko dla tablic zmiennych
  DBMS_OUTPUT.PUT_LINE('Obecnie jest: ' || litery.COUNT);
  DBMS_OUTPUT.PUT_LINE('Maksymalnie może być ' || litery.LIMIT);
  DBMS_OUTPUT.PUT_LINE('Pozostało wolnego: ' || (litery.LIMIT - litery.COUNT));

END;
/

/*************************************
WYNIK:

1
Brak 999
Liczba elementów: 3
Pierwszy i ostatni element: 1 10
```

```
1
4
77
Obecnie jest: 4
Maksymalnie może być 7
Pozostało wolnego: 3
**************************************/
```

Komentarz:

Termin *metody* został zaczerpnięty z technik obiektowych.

9. Dynamiczny SQL

9.1. Instrukcja EXECUTE IMMEDIATE

Przykład 46

```
DECLARE
  plecenie VARCHAR2(100) := 'DROP TABLE moja-tablela';
BEGIN
  EXECUTE IMMEDIATE polecenie;
END;

PL/SQL procedure successfully completed.
```

Komentarz:

Tabela zostanie naprawdę wykasowana. Język PL/SQL jako taki nie wpiera poleceń DDL (ang. *Data Definition Language*), jednak można je wykonywać korzystając z możliwości jakie daje dynamiczny SQL.

Przykład 47

```
BEGIN
  EXECUTE IMMEDIATE
  'CREATE TABLE tab_temp_' || :nazwa || '(id NUMBER(10), kolumna VARCHAR2(30))';

  EXECUTE IMMEDIATE
  'INSERT INTO tab_temp_' || :nazwa || ' VALUES (1, ''Gramacki'')';
END;
```

Komentarz:

Aby przekazać do bloku anonimowego jakieś wartości posługujemy się tzw. zmiennymi wiązanymi (ang. *bind variables*). Zmienne te tworzymy w programie SQL*Plus. Następnie mogą one być przekazane do programu PL/SQL. Szczegóły patrz dokumentacja programu SQL*Plus.

```
-- Definicja zmiennej. Nazwę polecenia można skrócić do VAR.
SQL> VARIABLE nazwa VARCHAR2(30);
```

```
-- Przypisanie zdefiniowanej zmiennej pewnej wartości
SQL> BEGIN :nazwa := 'moja_tabela'; END;
  2  /

PL/SQL procedure successfully completed.
```

```
-- Oglądamy aktualnie zdefiniowane zmienne.
SQL> VARIABLE
zmienna   id
typ danych   NUMBER

zmienna   nazwa
typ danych   VARCHAR2(30)
```

```
-- Sprawdzamy wartość zmiennej.
SQL> PRINT nazwa

NAZWA
--------------------------------
moja_tabela
```

```
-- Wykonujemy program PL/SQL.

SQL> BEGIN
  2    EXECUTE IMMEDIATE
  3    'CREATE TABLE tab_temp_' || :nazwa || '(id NUMBER(10), kolumna VARCHAR2(30))';
  4
  5    EXECUTE IMMEDIATE
  6    'INSERT INTO tab_temp_' || :nazwa || ' VALUES (1, ''Gramacki'')';
  7  END;
  8  /

PL/SQL procedure successfully completed.
```

```
-- Sprawdzamy wynik działania.
SQL> SELECT table_name FROM user_tables WHERE table_name LIKE 'TAB_TEMP%';

TABLE_NAME
------------------------------
TAB_TEMP_MOJA_TABELA

SQL>
```

Przykład 48

```
DECLARE
  stmt VARCHAR2(100) := 'SELECT * FROM emp';
BEGIN
  EXECUTE IMMEDIATE stmt;
END;

PL/SQL procedure successfully completed.
```

Komentarz:

Zadziała ale i tak nic nie zobaczymy na ekranie. Użycie W tej postaci `EXECUTE IMMEDIATE` jest więc dyskusyjne.

Przykład 49

```
DECLARE
  imie VARCHAR2(30);
  polecenie VARCHAR2(100) := 'SELECT first_name FROM emp WHERE id  = 1';
BEGIN
  EXECUTE IMMEDIATE polecenie INTO imie;
END;

Carmen

PL/SQL procedure successfully completed.
```

Komentarz:

Trzeba oczywiście zadbać, aby `SELECT` zwrócił dokładnie jeden wiersz.

9.2. Zmienne kursorowe

Przykład 50

Zmienne kursorowe (ang. *cursor variable*) – przykład BŁĘDNY.

```
DELCARE
  stmp_sql VARCHAR2(512) := 'SELECT * FROM emp';
  CURSOR c_emp IS stmt_sql   -- Niestety nie zadziała!
BEGIN
  FOR stmt_sql IN c_emp      -- Niestety nie zadziała!
  LOOP
  ...
  END LOOP;
END;
```

Przykład 51

Zmienne kursorowe – przykład POPRAWNY.

```
-- Zmienna kursorowa to WSKAŹNIK do kursora (czyli do tego miejsca w pamięci,
-- gdzie w rzeczywistości znajduje się kursor.
DECLARE

  -- Najpierw deklarujemy typ danych, wskazujący na kursor.
  TYPE t_refcur IS REF CURSOR;

  -- Deklarujemy zmienną kursorową o typie REF CURSOR.
  -- Zmienna kursorowa TO NIE TO SAMO co kursor!
  c_cur t_refcur;

  -- Deklarujemy zmienną rekordową.
  uv_emp emp%ROWTYPE;

BEGIN
  -- Otwieramy zmienną kursorową. W MOMENCIE OTWARCIA jest ona KOJARZONA
  -- z zapytaniem SELECT.
  OPEN c_cur FOR SELECT * FROM emp WHERE last_name LIKE 'M%';
  LOOP
    FETCH c_cur INTO uv_emp;
    EXIT WHEN c_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(uv_emp.first_name||' '||uv_emp.last_name);
  END LOOP;
  CLOSE c_cur;
END;
```

Komentarz:

Istotą używania zmiennych kursorowych jest to, że program ma możliwość **odsunięcia w czasie momentu powiązania instrukcji SQL z odpowiednim kursorem** (czyli *de facto* miejscem w pamięci zarezerwowanym na przechowanie wyniku zapytania). Dopiero w momencie wykonania programu następuje powiązanie kursora z zapytaniem.

W przykładzie użyliśmy zmiennych kursorowych trochę „na siłę”, gdyż taki sam efekt można uzyskać stosując poprzednio poznane klasyczne kursory statyczne.

Przykład 52

Zmienne kursorowe – przykład 2. Tym razem procedura.

```
CREATE OR REPLACE PROCEDURE select_by_user (select_statement IN VARCHAR2) IS
-- Najpierw deklarujemy typ danych, wskazujący na kursor.
```

```
TYPE t_refcur IS REF CURSOR;

-- Można by też zrobić nieco bezpieczniej:
-- RETURN emp%ROWTYPE specyfikuje typ, jaki musi zwracać dowolny
-- kursor przyporządkowany do zmiennej kursorowej REF CURSOR.
-- TYPE t_refcur IS REF CURSOR RETURN emp%ROWTYPE;

-- Deklarujemy zmienną kursorową.
c_cur t_refcur;

-- Deklarujemy zmienną rekordową.
uv_emp emp%ROWTYPE;

BEGIN
  OPEN c_cur FOR select_statement;
  LOOP
    FETCH c_cur INTO uv_emp;
    EXIT WHEN c_cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(uv_emp.last_name);
  END LOOP;
  CLOSE c_cur;
END;
```

Przykład 53

```
CREATE OR REPLACE PROCEDURE select_by_user
  (tabela VARCHAR2, kolumna VARCHAR2, minmax IN VARCHAR2) IS

TYPE t_refcur IS REF CURSOR;
c_cur t_refcur;
zapytanie VARCHAR2(1000);
wynik NUMBER;

BEGIN
  -- Tworzymy zapytanie typu: 'select max(id) from emp'
  zapytanie := 'SELECT ' || minmax || '(' || kolumna || ') FROM ' || tabela;

  -- Zapytanie zawsze będzie zwracało jeden wiesz, więc można tak jak niżej.
  EXECUTE IMMEDIATE zapytanie INTO wynik;
  DBMS_OUTPUT.PUT_LINE(wynik);
END;
```

Komentarz:

Używając dynamicznego SQL-a możemy praktycznie dowolnie budować zapytanie. Poniżej pokazano przykładowe dwa wywołania powyższej procedury:

```
SQL> EXECUTE select_by_user ('emp', 'id', 'max');
25

PL/SQL procedure successfully completed.

SQL> EXECUTE select_by_user('inventory', 'product_id', 'min');
10011

PL/SQL procedure successfully completed.
```

Przykład 54

```
CREATE OR REPLACE PROCEDURE select_by_user (dept_id NUMBER) IS

TYPE t_refcur IS REF CURSOR;
c_cur t_refcur;
```

```
uv_imie VARCHAR2(20);
uv_nazw VARCHAR2(30);
uv_zar  NUMBER;
zapytanie VARCHAR2(100);

BEGIN

  zapytanie :=
    'SELECT first_name, last_name, salary ' ||
    'FROM emp WHERE dept_id = ' || dept_id;

  OPEN c_cur FOR zapytanie;
  LOOP
     FETCH c_cur INTO uv_imie, uv_nazw, uv_zar;
     EXIT WHEN c_cur%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE(uv_imie ||', '|| uv_nazw ||', '|| uv_zar);
  END LOOP;
  CLOSE c_cur;
END;
```

Komentarz:

Przykładowy wynik:

```
SQL> EXECUTE  select_by_user (41);
LaDoris, Ngao, 1450
Molly, Urguhart, 1200
Elena, Maduro, 1400
George, Smith, 940

PL/SQL procedure successfully completed.
```

9.3. Pewien bardziej złożony przykład użycia zmiennych kursorowych

Przykład 55

Definiujemy pakiet (informacje na temat pakietów – patrz wcześniejsze rozdziały).

```
CREATE OR REPLACE PACKAGE ref_cursor_sample AS

  -- Definiujemy na poziome pakietu.
  TYPE t_refcur IS REF CURSOR;

  PROCEDURE open_cur_variable (in_refcur IN OUT t_refcur, in_tbl IN NUMBER);
END ref_cursor_sample;
```

Definiujemy ciało pakietu.

```
CREATE OR REPLACE PACKAGE BODY ref_cursor_sample AS

PROCEDURE open_cur_variable (in_refcur IN OUT t_refcur, in_tbl IN NUMBER) IS
BEGIN

 IF in_tbl = 1 THEN
    OPEN in_refcur FOR
       SELECT *
       FROM emp
       WHERE dept_id < 40
       ORDER BY dept_id, last_name;

  ELSIF in_tbl = 2 THEN
    OPEN in_refcur FOR
```

```
       SELECT D.name, D.id, R.name
       FROM dept D, region R
       WHERE D.region_id = R.id AND
             R.id = 1
       ORDER BY D.id;
  END IF;

END open_cur_variable;

-- Koniec definicji ciała pakietu.
END ref_cursor_sample;
```

Używamy utworzoną wcześniej procedurę.

```
DECLARE
  -- Deklarujemy zmienną kursorową zdefiniowaną w specyfikacji pakietu.
  -- Taka zmienna jest dostępna dla każdego klienta PL/SQL - w tym
  -- przykładzie używamy jej w bloku anonimowym PL/SQL.
  c_refcur ref_cursor_sample.t_refcur;
  uv_dn dept.name%TYPE;
  uv_rn region.name%TYPE;
  uv_did dept.id%TYPE;
  uv_emp emp%ROWTYPE;
BEGIN
  -- Wywołujemy procedurę otwierającą zmienną kursorową.
  ref_cursor_sample.open_cur_variable(c_refcur, 1);
  DBMS_OUTPUT.PUT_LINE('---------------- tabela EMP ----------------');
  LOOP
    FETCH c_refcur INTO uv_emp;
      EXIT WHEN c_refcur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(
        RPAD(uv_emp.first_name||' '||uv_emp.last_name,35)||uv_emp.dept_id);
  END LOOP;
  -- Zamknięcie zmiennej kursorowej.
  CLOSE c_refcur;

  -- Drugi raz wywołujemy procedurę otwierającą zmienną kursorową.
  ref_cursor_sample.open_cur_variable(c_refcur, 2);
  DBMS_OUTPUT.PUT_LINE('--------------- tabela DEPT ----------------');
  LOOP
    FETCH c_refcur INTO uv_dn, uv_did, uv_rn;
    EXIT WHEN c_refcur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(RPAD(uv_dn||', '||uv_rn, 35)||uv_did);
  END LOOP;
  CLOSE c_refcur;

END;
```

Komentarz:

W wyniku wykonania powyższego kodu otrzymujemy taki wynik:

```
---------------- tabela EMP ----------------
Mark Quick-To-See                  10
Colin Magee                        31
Midori Nagayama                    31
Henry Giljum                       32
Yasmin Sedeghi                     33
Mai Nguyen                         34
Radha Patel                        34
Andre Dumas                        35
--------------- tabela DEPT ----------------
Finance, North America             10
```

```
Sales, North America                31
Operations, North America           41
Administration, North America       50
```

9.4. Dynamiczny SQL z wykorzystaniem pakietu DBMS_SQL

Do wersji 8 bazy danych Oracle nie było możliwe używanie dynamicznego SQL-a w takiej postaci jak to pokazano wyżej. Jedynym sposobem tworzenia dynamicznych wyrażeń SQL był pakiet `DBMS_SQL` (pakiet ten jest nadal dostępny we wszystkich wersjach Oracle, łącznie z tymi najnowszymi). Jego używanie jest jednak dużo mniej wygodne niż prezentowana tu metoda. Równoważne kody są dużo większe i mniej czytelne. Ponadto programy wykorzystujące pakiet `DBMS_SQL` zwykle wykonują się nieco wolniej. Więcej szczegółów można znaleźć w dokumentacji.

10. Wyzwalacze bazodanowe

10.1. Składnia (uproszczona)

```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza
   {BEFORE | AFTER} zdarzenie_wywołujace ON nazwa_tabeli
[REFERENCING {NEW AS | OLD AS} nazwa_kwalifikatora}
[FOR EACH ROW]
[WHEN (wyrażenie)]
[DECLARE
  ...sekcja deklaracji zawierająca wszystkie lokalne typy, zmienne,
     stałe, kursory i deklaracje podprogramów.]
BEGIN
  ...sekcja instrukcji
[EXCEPTION
  ...sekcja obsługi wyjątków]
END;
```

10.2. Możliwe typy wyzwalaczy

```
row triggers
------------
Trigger fires once for each row of the table that is
affected by the triggering statement.

BEFORE INSERT (UPDATE, DELETE) FOR EACH ROW
 AFTER INSERT (UPDATE, DELETE) FOR EACH ROW

statement triggers
------------------
The absence of the FOR EACH ROW option indicates that the trigger fires only
once for each applicable statement, but not separately for each row affected by
the statement.

BEFORE INSERT (UPDATE, DELETE)
 AFTER INSERT (UPDATE, DELETE)
```

10.3. Przykłady

Przykład 56

Jeden z najprostszych wyzwalaczy.

```
CREATE OR REPLACE TRIGGER dept_t1
BEFORE INSERT OR UPDATE OR DELETE ON dept

BEGIN
  IF INSERTING THEN
    DBMS_OUTPUT.PUT_LINE('before insert trigger fired');
  ELSIF UPDATING THEN
    DBMS_OUTPUT.PUT_LINE('before update trigger fired');
  ELSIF DELETING THEN
    DBMS_OUTPUT.PUT_LINE('before delete trigger fired');
  END IF;
END;
```

Przykład 57

Wyzwalacz tylko dla wybranych kolumn.

```
-- Klauzula OF możliwa jest tylko dla wyzwalacza UPDATE.

CREATE OR REPLACE TRIGGER dept_t1
AFTER UPDATE OF first_name, last_name ON demp

BEGIN
    DBMS_OUTPUT.PUT_LINE('Kolumny zostały zmodyfikowane.');
  END IF;
END;
```

Przykład 58

Implementacja modułu auditingu.

```
DROP SEQUENCE dept_aud_seq;
DROP TABLE dept_aud;

-- Tworzymy tabelę oparta o DEPT, z tym że dodajemy kilka
-- kolumn „systemowych”. Są one wytłuszczone.
CREATE TABLE dept_aud (
  id_aud     NUMBER PRIMARY KEY,     -- kolumna "auditingowa"
  id         NUMBER(7),              -- | ================================
  name       VARCHAR2(25),           -- | oryginalne kolumny z tabeli dept
  region_id  NUMBER(7),              -- | ================================
  data       DATE,                   -- kolumna "auditingowa"
  osoba      VARCHAR2(30),           -- kolumna "auditingowa"
  typ        CHAR(1)                 -- kolumna "auditingowa"
);

CREATE SEQUENCE dept_aud_seq;
```

```
CREATE OR REPLACE TRIGGER dept_t2
BEFORE INSERT OR UPDATE OR DELETE ON DEPT
FOR EACH ROW
BEGIN
  IF INSERTING THEN
    INSERT INTO dept_aud
    VALUES(s_dept_aud_seq.NEXTVAL,
          :NEW.id, :NEW.name, :NEW.region_id, sysdate, user, 'I');

  ELSIF UPDATING THEN
    INSERT INTO dept_aud
    VALUES(s_dept_aud_seq.NEXTVAL,
          :OLD.id, :OLD.name, :OLD.region_id, sysdate, user, 'U');
    INSERT INTO dept_aud
```

```
    VALUES(s_dept_aud_seq.NEXTVAL,
          :NEW.id, :NEW.name, :NEW.region_id, sysdate, user, 'U');

  ELSIF DELETING THEN
    INSERT INTO dept_aud
    VALUES(s_dept_aud_seq.NEXTVAL,
           :OLD.id, :OLD.name, :OLD.region_id, sysdate, user, 'D');

  END IF;
END;
```

10.4. Włączanie, wyłączanie i kasowanie, kompilowanie wyzwalaczy

```
ALTER TRIGGER dept_t2 DISABLE;
ALTER TRIGGER dept_t2 ENABLE;

ALTER TABLE dept ENABLE ALL TRIGGERS;
ALTER TABLE dept DISABLE ALL TRIGGERS;

ALTER TRIGGER dept_t2 COMPILE;

DROP TRIGGER dept_t2;
```

10.5. Wyświetlanie informacji na temat wyzwalaczy

```
SQL> DESC USER_TRIGGERS

TRIGGER_NAME                   VARCHAR2(30)
TRIGGER_TYPE                   VARCHAR2(16)
TRIGGERING_EVENT               VARCHAR2(216)
TABLE_OWNER                    VARCHAR2(30)
BASE_OBJECT_TYPE               VARCHAR2(16)
TABLE_NAME                     VARCHAR2(30)
COLUMN_NAME                    VARCHAR2(4000)
REFERENCING_NAMES              VARCHAR2(128)
WHEN_CLAUSE                    VARCHAR2(4000)
STATUS                         VARCHAR2(8)
DESCRIPTION                    VARCHAR2(4000)
ACTION_TYPE                    VARCHAR2(11)
TRIGGER_BODY                   LONG  -- uwaga na ten typ danych!
```

```
SELECT
  table_name, trigger_name, trigger_type, triggering_event,
  referencing_names, status, action_type
FROM
  USER_TRIGGERS;
```

TABLE_NAME	S_DEPT	S_DEPT
TRIGGER_NAME	S_DEPT_T1	S_DEPT_T2
TRIGGER_TYPE	BEFORE STATEMENT	BEFORE EACH ROW
TRIGGERING_EVENT	INSERT OR UPDATE OR DELETE	INSERT OR UPDATE OR DELETE
REFERENCING_NAMES	REFERENCING NEW AS NEW OLD AS OLD	REFERENCING NEW AS NEW OLD AS OLD
STATUS	ENABLED	ENABLED
ACTION_TYPE	PL/SQL	PL/SQL

```
SQL> SELECT trigger_body FROM user_triggers;
```

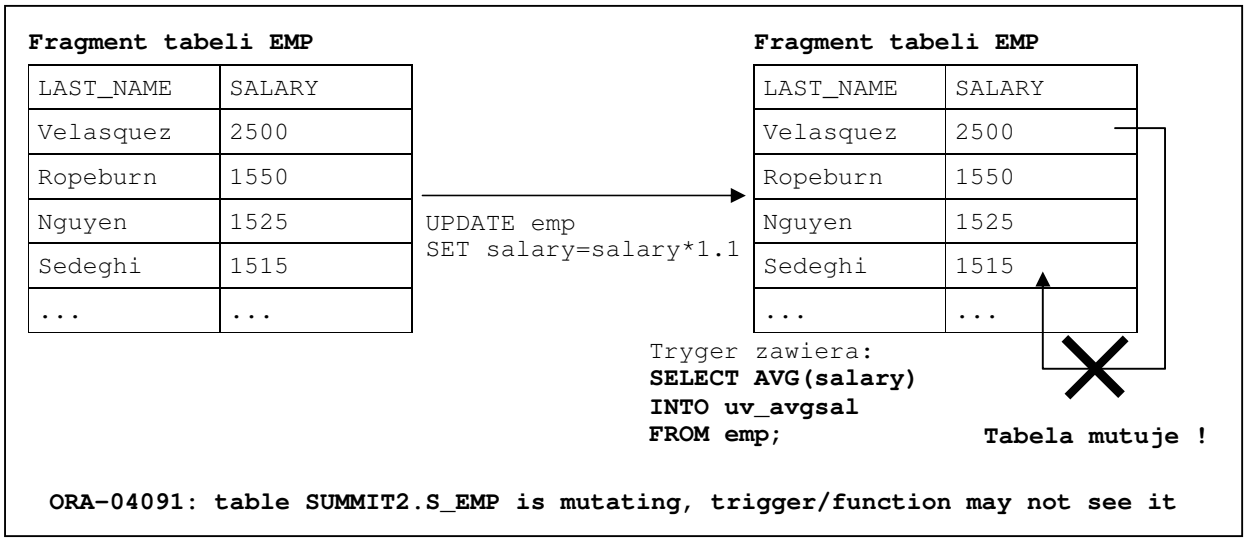
```
Nie wyświetlił się cały kod trygera. "Winna" jest tutaj zmienna LONG programu
SQL*Plus. Domyślnie jej wartość jest ustawiana na 80. Szczegóły patrz
dokumentacja.

TRIGGER_BODY
--------------------------------------------------------------------------------
BEGIN
  IF INSERTING THEN
    DBMS_OUTPUT.PUT_LINE('before insert trigger fired'

Potwierdzamy to co napisano powyżej.

SQL> show long
long 80

Ustawiamy więc zmienną na większą wartość.

SQL> set long 2000

Teraz kod trygera wyświetla się w całości.

SQL> SELECT trigger_body FROM user_triggers;

TRIGGER_BODY
--------------------------------------------------------------------------------
BEGIN
  IF INSERTING THEN
    DBMS_OUTPUT.PUT_LINE('before insert trigger fired');
  ELSIF UPDATING THEN
    DBMS_OUTPUT.PUT_LINE('before update trigger fired');
  ELSIF DELETING THEN
    DBMS_OUTPUT.PUT_LINE('before delete trigger fired');
  END IF;
END;
```

10.6. Mutujące tablice (ang. *mutating tables*)

Przykład 59

Niech na tabeli EMP będzie założony następujący wyzwalacz:

```
CREATE OR REPLACE TRIGGER emp_avgsal
AFTER UPDATE ON emp
FOR EACH ROW
DECLARE
    uv_avgsal NUMBER;
BEGIN
    SELECT AVG(salary) INTO uv_avgsal FROM emp;
    DBMS_OUTPUT.PUT_LINE(' Srednie zarobki: ' || uv_avgsal);
END;
```



Uwagi:

- Problem związany z mutującymi tabelami występuje TYLKO w przypadku trygerów wierszowych (*row level triggers*), czyli tych zawierających opcję `FOR EACH ROW`.
- W naszym przykładzie niemożliwe jest obliczenie średniej zarobków, gdy w tym samym czasie modyfikowane są wartości tych zarobków. Średnią tą będzie można wyliczyć dopiero PO uaktualnieniu zarobków a nie w trakcie uaktualniania!!!
- Gdy usuniemy z definicji wyzwalacza opcję `FOR EACH ROW` zadziała on poprawnie (inaczej: przekształcimy wyzwalacz wierszowy (*row trigger*s) w wyzwalacz poleceniowy (*statement triggers*).
- Obejście problemu mutujących tabel jest dość niewygodne w praktyce (trzeba używać tabel tymczasowych tabel PL/SQL lub zmiennych pakietowych – patrz dokumentacja).
- W podanym tu przykładzie tak naprawdę nie jest potrzeby wyzwalacz typu *row level*, gdyż do obliczenia średnich zarobków w tabeli `EMP` nie ma potrzeby uruchamiać wyzwalacza dla każdej przetwarzanej kolumny. Podany przykład należy więc traktować tylko jako omówienie problemu mutujących tabel.

10.7. Przykład z jedną tabelą

Źródło: http://www.databasejournal.com/features/oracle/article.php/3329121

Problem związany z mutującymi tabelami może wystąpi także przy pracy z JEDNĄ tabelą.

```
Case 1: When Trigger on table refers the same table:
-------------------------------------------------------------
OPERATION       TYPE                          MUTATING?
-------------------------------------------------------------
insert          before/statement-level        No
insert          after/statement-level         No
update          before/statement-level        No
update          after/statement-level         No
delete          before/statement-level        No
delete          after/statement-level         No

insert          before/row-level              Single row    Multi-row
                                              No            Yes
insert          after/row-level               Yes
```

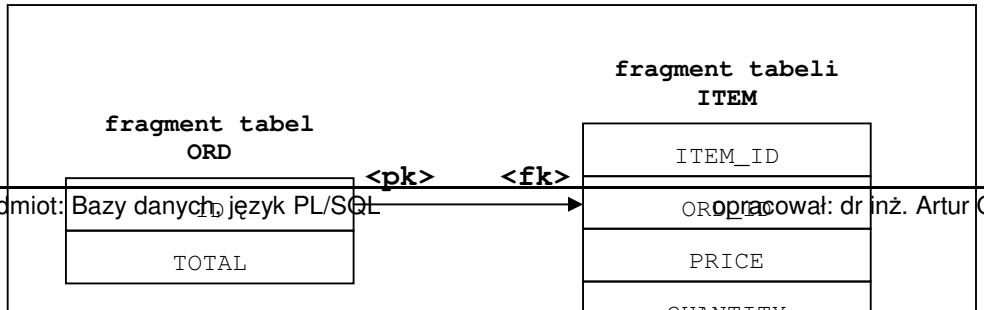
```
update          before/row-level            Yes
update          after/row-level             Yes
delete          before/row-level            Yes
delete          after/row-level             Yes
-------------------------------------------------------------
```

```
CREATE TABLE Tab
  (col1  NUMBER,
   col2  VARCHAR2(30));

Table created.

CREATE OR REPLACE TRIGGER Tab_trg
  BEFORE INSERT OR UPDATE OR DELETE ON Tab FOR EACH ROW
  DECLARE
    suma pls_integer;
  BEGIN
    SELECT count(1) INTO suma FROM Tab;
   -- more processing...
 END;

Trigger created.

INSERT INTO Tab VALUES (1, 'testing');

1 row created.

UPDATE Tab SET col1 = 2;

-- Komunikat o błędzie

UPDATE TAB SET col1 = 2;
       *
ERROR at line 1:
ORA-04091: table SYSTEM.TAB is mutating, trigger/function may not see it
ORA-06512: at "SYSTEM.TAB_TRG", line 4
ORA-04088: error during execution of trigger 'SYSTEM.TAB_TRG'
```

10.8. Bardziej złożony przykład z tabelami mutującymi

Rozważmy dwie tabele z modelu SUMMIT2: ORD oraz ITEM. W tabeli ORD występuje kolumna TOTAL natomiast w tabeli ITEM występują kolumny PRICE oraz QUANTITY (patrz rysunek poniżej).

Zgodnie z logiką modelu SUMMIT2 mamy, że kolumna ORD.TOTAL zawiera (powinna zawierać) sumaryczną wartość sprzedaży na danym zamówieniu. Jest ona ilościowo równa wartości wszystkich zamówionych produktów, których szczegóły zapisane są w tabeli ITEM (czyli jest to suma wszystkich iloczynów ITEM.PRICE * ITEM.QUANTITY dla danego zamówienia).



Poniższe polecenia SELECT pokazują szczegóły zamówienia o numerze 100

```
SELECT total FROM ord WHERE id=100;

TOTAL
------
601100


SELECT price, quantity, price*quantity "price*quantity"
FROM item WHERE ord_id=100;

PRICE QUANTITY price*quantity
----- -------- --------------
135   500      67500
380   400      152000
14    500      7000
582   600      349200
8     250      2000
20    450      9000
36    400      14400


SELECT SUM(price*quantity) "total" FROM item WHERE ord_id=100;

TOTAL
------
601100
```

Sam model relacyjny nie jest w stanie zapewnić właściwej synchronizacji kolumn w tabelach ORD oraz ITEM. Należy więc napisać wyzwalacz, który zsynchronizuje wartości w tych trzech kolumnach. Ma to działać w taki sposób, że po jakiejkolwiek zmianie ceny (S_ITEM.PROCE) i/lub ilości sprzedaży (S_ITEM.QUANTITY) danego produktu, **automatycznie** zostanie uaktualniona kolumna ORD.TOTAL.

Tryger jak poniżej nie zadziała prawidłowo, gdyż pojawi się błąd **ORA-04091** (mutowanie tabel).

```
CREATE OR REPLACE TRIGGER item_count_total
AFTER INSERT OR UPDATE OR DELETE OF price, quantity ON item FOR EACH ROW

-- PRZYKŁAD BŁĘDNY !!!

DECLARE
  CURSOR c_cur IS
  SELECT price, quantity
  FROM item
  WHERE ord_id = :new.ord_id;

  uv_price item.price%TYPE;
  uv_quantity item.quantity%TYPE;
  uv_total ord.total%TYPE;
```

```
BEGIN
  OPEN c_cur;

  -- Liczymy sumę wydatków: price*quantity
  LOOP
    FETCH c_cur INTO uv_price, uv_quantity;
      EXIT WHEN c_cur%NOTFOUND;
    uv_total := uv_total + uv_price*uv_quantity;
  END LOOP;

  -- Wyliczoną sumę wpisujemy do tabeli ORD
  -- Tu pojawi się ORA-04091
  UPDATE ord SET total = uv_total WHERE id = :new.ord_id;
  CLOSE c_cur;
END;
/
```

Trzeba w tym przypadku postąpić w inny sposób. Należy wymyślić metodę, która pozwoli wykonać działania przewidziane dla trygera typu *row-level* (czyli `FOR EARCH ROW`) przez „bezpieczniejszy" w tym kontekście tryger typu *statement*.

Aby to osiągnąć użyjemy tymczasowej **tabeli PL/SQL**, której zadaniem będzie przechowanie „na chwilę" tych rekordów (a w zasadzie tylko ich unikalnych numerów *rowid*), które są modyfikowane. Później te dane zostaną wykorzystane przez tryger typu *statement* do wykonania zamierzonego zadania. Oto jak wygląda to w praktyce:

Tworzymy pakiet, w którym definiujemy **dwie tabele PL/SQL.** Jedna będzie służyła do przechowywania numerów *rowid* (tabela `new_rows`). Druga (tabela `empty`) będzie zawsze pusta i będzie służyła nam do „zerowania" pierwszej tabeli.

```
CREATE OR REPLACE PACKAGE state_pkg AS
  -- Definiujemy tzw. kolekcje (rodzaj tablicy).
  -- Szczegóły patrz:
  -- PL/SQL User's Guide and Reference, rozdział: Collections and Records

  -- Patrz też artykuł "Avoiding Mutating Tables"
  -- http://osi.oracle.com/~tkyte/Mutate/index.html

  TYPE rowid_array IS TABLE OF ROWID INDEX BY BINARY_INTEGER;
  new_rows rowid_array;
  empty rowid_array;
END state_pkg;
/
```

Krok 1:

W pierwszym kroku tworzymy wyzwalacz `BEFORE`, który „wyzeruje" nam tabelę PL/SQL.

```
CREATE OR REPLACE TRIGGER item_bi
BEFORE INSERT OR UPDATE OR DELETE ON item
BEGIN
  state_pkg.new_rows := state_pkg.empty;
END;
/
```

Krok 2:

W drugim kroku tworzymy wyzwalacz `AFTER` typy *row-level*. Jego zadaniem jest zapisanie w tabeli PL/SQL wszystkich unikalnych numerów *rowid*. Będzie to nasza „przechowalnia", z której skorzysta następny tryger.

```
CREATE OR REPLACE TRIGGER item_aifer
AFTER INSERT OR UPDATE OR DELETE OF price, quantity ON item FOR EACH ROW
BEGIN
  state_pkg.new_rows (state_pkg.new_rows.count + 1) := :new.rowid;
END;
/
```

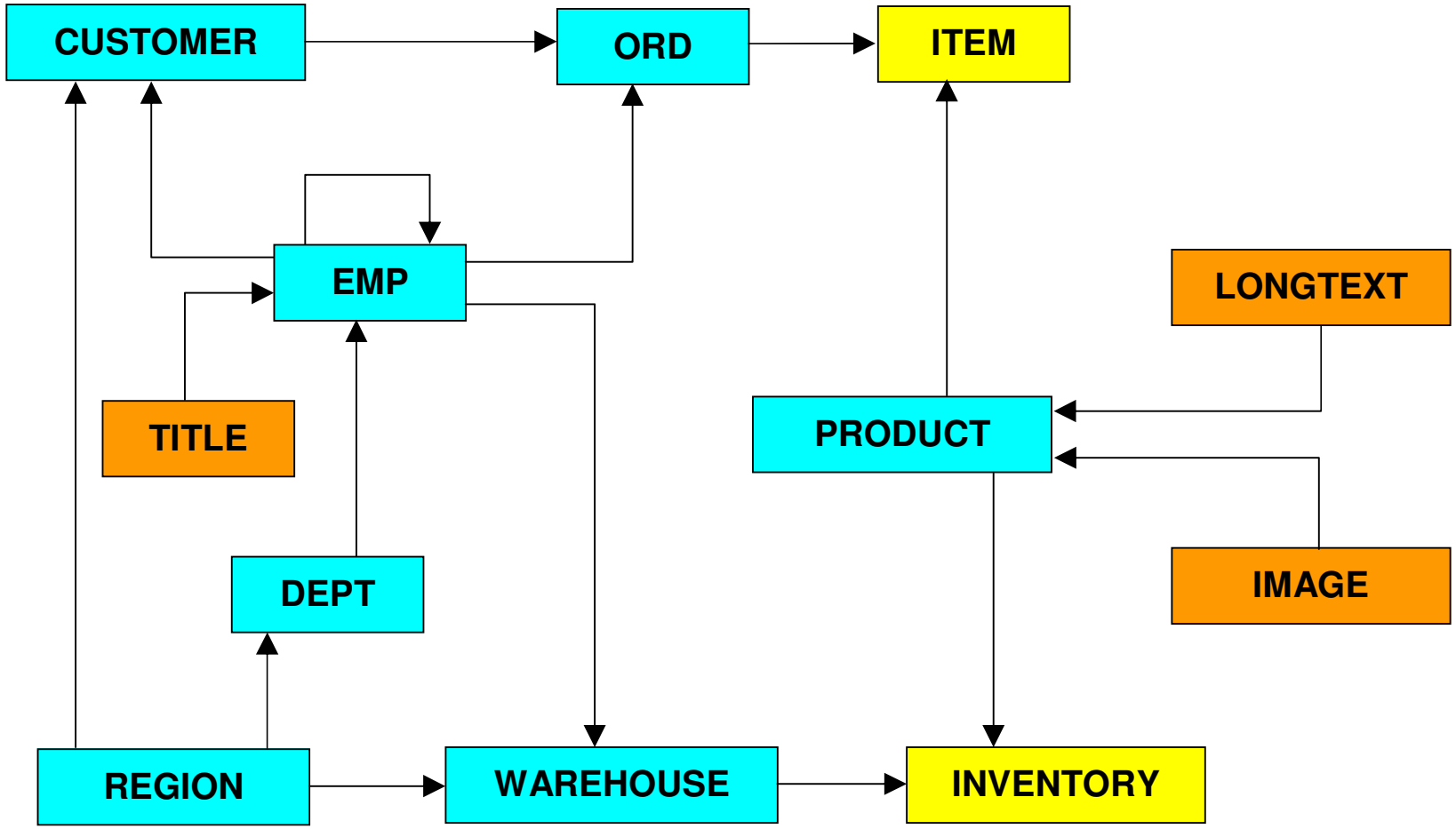
Krok 3:

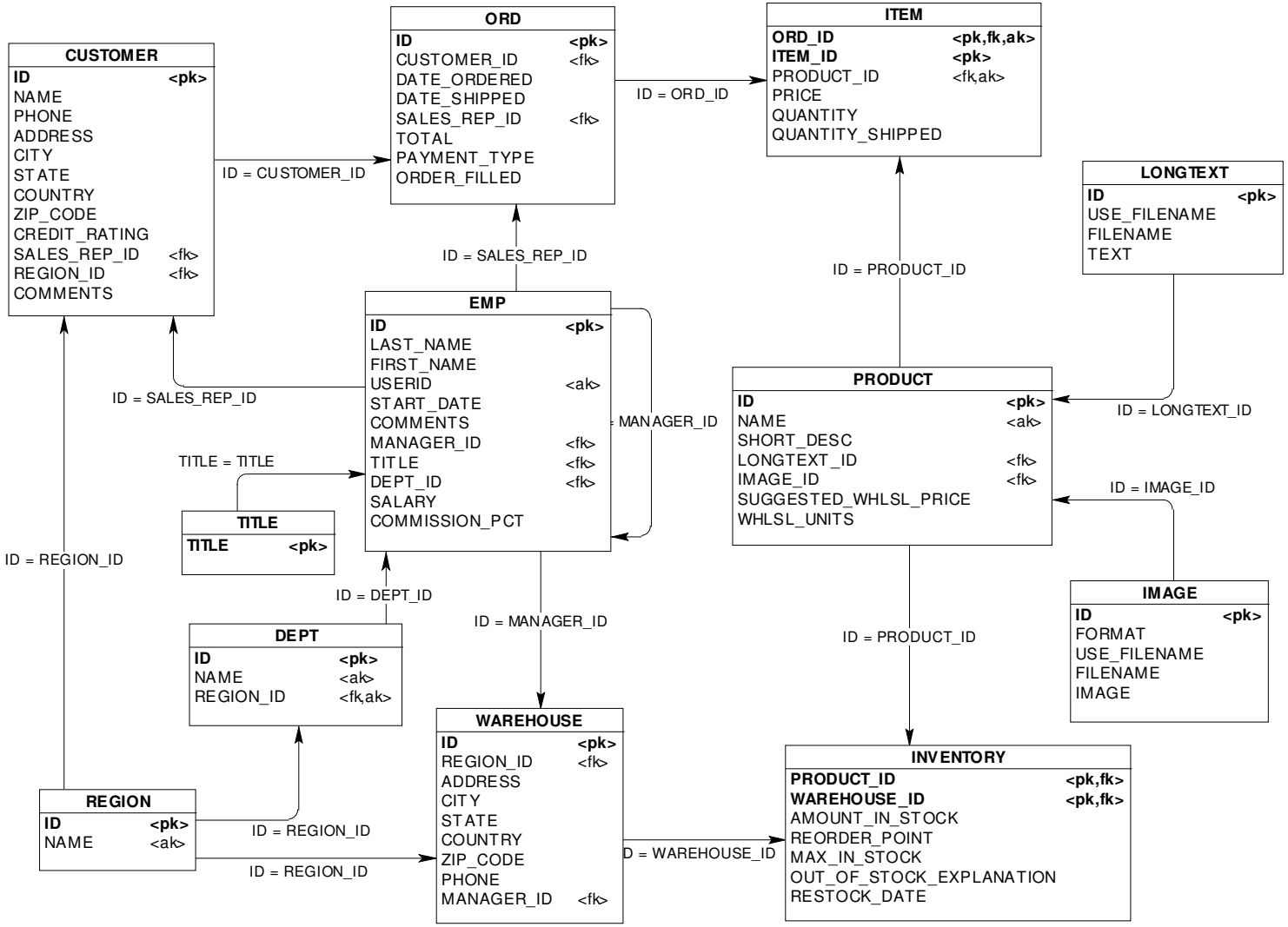
W trzecim kroku tworzymy wyzwalacz `AFTER` typy *statement*. Jego zadaniem jest pobranie danych z tabeli PL/SQL (numerów *rowid*) a następnie na ich podstawie wykonanie zadanego zadania (uaktualnienie kolumny `ORD.TOTAL`). Ponieważ tryger jest typu *statement*, więc na pewno nie pojawi się problem mutujących tabel.

```
CREATE OR REPLACE TRIGGER item_ai
AFTER INSERT OR UPDATE OR DELETE OF price, quantity ON item

DECLARE
  CURSOR c_cur (in_ord_id item.ord_id%TYPE) IS
  SELECT price, quantity
  FROM item
  WHERE ord_id = in_ord_id;

  uv_ord_id item.ord_id%TYPE;
  uv_price item.price%TYPE;
  uv_quantity item.quantity%TYPE;
  uv_total ord.total%TYPE := 0;

BEGIN
 -- Odczytaj dane z tabeli PL/SQL
 FOR i IN 1 .. state_pkg.new_rows.count LOOP
   -- Pobierz numer zamówienia (ord_id) i zapisz go do zmiennej.
   SELECT ord_id INTO uv_ord_id FROM item WHERE rowid = state_pkg.new_rows(i);

   -- Otwórz kursor i wylicz wielkość całkowitej sprzedaży (uv_total).
   OPEN c_cur(uv_ord_id);
   LOOP
     FETCH c_cur INTO uv_price, uv_quantity;
     EXIT WHEN c_cur%NOTFOUND;
     uv_total := uv_total + uv_price*uv_quantity;
   END LOOP;
   CLOSE c_cur;

    -- Uaktualnij odpowiedni rekord w tabeli ord.
    UPDATE ord SET total = uv_total WHERE id = uv_ord_id;

    -- Zresetuj przed nowym obrotem pętli.
    uv_total :=0;

  END LOOP;
END;
```

11. Model SUMMIT2

11.1. Omówienie




CUSTOMER
ID <pk>
NAME
PHONE
ADDRESS
CITY
STATE
COUNTRY
ZIP_CODE
CREDIT_RATING
SALES_REP_ID <fk>
REGION_ID <fk>
COMMENTS
ID = CUSTOMER_ID
ORD
ID <pk>
CUSTOMER_ID <fk>
DATE_ORDERED
DATE_SHIPPED
SALES_REP_ID <fk>
TOTAL
PAYMENT_TYPE
ORDER_FILLED
ID = ORD_ID
ITEM
ORD_ID <pk,fk,ak>
ITEM_ID <pk>
PRODUCT_ID <fk,ak>
PRICE
QUANTITY
QUANTITY_SHIPPED
LONGTEXT
ID <pk>
USE_FILENAME
FILENAME
TEXT
ID = SALES_REP_ID
ID = PRODUCT_ID
ID = LONGTEXT_ID
EMP
ID <pk>
LAST_NAME
FIRST_NAME
USERID <ak>
START_DATE
COMMENTS
MANAGER_ID <fk>
TITLE <fk>
DEPT_ID <fk>
SALARY
COMMISSION_PCT
ID = SALES_REP_ID
= MANAGER_ID
PRODUCT
ID <pk>
NAME <ak>
SHORT_DESC
LONGTEXT_ID <fk>
IMAGE_ID <fk>
SUGGESTED_WHLSL_PRICE
WHLSL_UNITS
ID = IMAGE_ID
TITLE = TITLE
TITLE
TITLE <pk>
ID = REGION_ID
ID = DEPT_ID
ID = MANAGER_ID
ID = PRODUCT_ID
IMAGE
ID <pk>
FORMAT
USE_FILENAME
FILENAME
IMAGE
DEPT
ID <pk>
NAME <ak>
REGION_ID <fk,ak>
WAREHOUSE
ID <pk>
REGION_ID <fk>
ADDRESS
CITY
STATE
COUNTRY
ZIP_CODE
PHONE
MANAGER_ID <fk>
INVENTORY
PRODUCT_ID <pk,fk>
WAREHOUSE_ID <pk,fk>
AMOUNT_IN_STOCK
REORDER_POINT
MAX_IN_STOCK
OUT_OF_STOCK_EXPLANATION
RESTOCK_DATE
REGION
ID <pk>
NAME <ak>
ID = REGION_ID
ID = REGION_ID
D = WAREHOUSE_ID

ID	NAME	PHONE	ADDRESS	CITY	STATE	COUNTRY	ZIP_CODE	CREDIT_RATING	SALES_REP_ID	REGION_ID	COMMENTS
201	Unisports	55-2066101	72 Via Bahia	Sao Paolo		Brazil		EXCELLENT	12	2	
202	OJ Atheletics	81-20101	6741 Takashi Blvd.	Osaka		Japan		POOR	14	4	
203	Delhi Sports	91-10351	11368 Chanakya	New Delhi		India		GOOD	14	4	
204	Womansport	1-206-104-0103	281 King Street	Seattle	Washington	USA		EXCELLENT	11	1	
205	Kam's Sporting Goods	852-3692888	15 Henessey Road	Hong Kong				EXCELLENT	15	4	
206	Sportique	33-2257201	172 Rue de Rivoli	Cannes		France		EXCELLENT	15	5	
207	Sweet Rock Sports	234-6036201	6 Saint Antoine	Lagos		Nigeria		GOOD		3	
208	Muench Sports	49-527454	435 Gruenestrasse	Stuttgart		Germany		GOOD	15	5	
209	Beisbol Si!	809-352689	792 Playa Del Mar	San Pedro de Macon's		Dominican Republic		EXCELLENT	11	1	
210	Futbol Sonora	52-404562	3 Via Saguaro	Nogales		Mexico		EXCELLENT	12	2	
211	Kuhn's Sports	42-111292	7 Modrany	Prague		Czechoslovakia		EXCELLENT	15	5	
212	Hamada Sport	20-1209211	57A Corniche	Alexandria		Egypt		EXCELLENT	13	3	
213	Big John's Sports Emporium	1-415-555-6281	4783 18th Street	San Francisco	CA	USA		EXCELLENT	11	1	
214	Ojibway Retail	1-716-555-7171	415 Main Street	Buffalo	NY	USA		POOR	11	1	
215	Sporta Russia	7-3892456	6000 Yekatamina	Saint Petersburg		Russia		POOR	15	5	

CUSTOMER

ORD_ID	ITEM_ID	PRODUCT_ID	PRICE	QUANTITY	QUANTITY_SHIPPED
97	1	20106	9	1000	1000
100	1	10011	135	500	500
100	2	10013	380	400	400
100	3	10021	14	500	500
100	4	10023	36	400	400
102	1	20108	28	100	100
106	1	20108	28	46	46
107	1	20106	11	50	50
107	2	20108	28	22	22
109	1	10011	140	150	150
109	2	10012	175	600	600
109	3	10022	21,95	300	300
112	1	20106	11	50	50

Uwaga: pokazano tylko pozycje zawierające produkty, gdzie **PRODUCT_ID** jest od 10011 do 20108

ITEM

Uwaga: pokazano tylko produkty, gdzie **ID** jest od 10011 do 20108

ID	NAME	SHORT_DESC	LONGTEXT_ID	IMAGE_ID	SUGGESTED_WHLSL_PRICE	WHLSL_UNITS
10011	Bunny Boot	Beginner's ski boot	518	1001	150	
10012	Ace Ski Boot	Intermediate ski boot	519	1002	200	
10013	Pro Ski Boot	Advanced ski boot	520	1003	410	
10021	Bunny Ski Pole	Beginner's ski pole	528	1011	16,25	
10022	Ace Ski Pole	Intermediate ski pole	529	1012	21,95	
10023	Pro Ski Pole	Advanced ski pole	530	1013	40,95	
20106	Junior Soccer Ball	Junior soccer ball	613		11	
20108	World Cup Soccer Ball	World cup soccer ball	615		28	

PRODUCT

ID	CUSTOMER_ID	DATE_ORDERED	DATE_SHIPPED	SALES_REP_ID	TOTAL	PAYMENT_TYPE	ORDER_FILLED
100	204	1992-08-31	1992-09-10	11	601100	CREDIT	Y
101	205	1992-08-31	1992-09-15	14	8056,6	CREDIT	Y
102	206	1992-09-01	1992-09-08	15	8335	CREDIT	Y
103	208	1992-09-02	1992-09-22	15	377	CASH	Y
104	208	1992-09-03	1992-09-23	15	32430	CREDIT	Y
105	209	1992-09-04	1992-09-18	11	2722,24	CREDIT	Y
106	210	1992-09-07	1992-09-15	12	15634	CREDIT	Y
107	211	1992-09-07	1992-09-21	15	142171	CREDIT	Y
108	212	1992-09-07	1992-09-10	13	149570	CREDIT	Y
109	213	1992-09-08	1992-09-28	11	1020935	CREDIT	Y
110	214	1992-09-09	1992-09-21	11	1539,13	CASH	Y
111	204	1992-09-09	1992-09-21	11	2770	CASH	Y
97	201	1992-08-28	1992-09-17	12	84000	CREDIT	Y
98	202	1992-08-31	1992-09-10	14	595	CASH	Y
99	203	1992-08-31	1992-09-18	14	7707	CREDIT	Y
112	210	1992-08-31	1992-09-10	12	550	CREDIT	Y

ORD

CUSTOMER

EMP

ID	REGION_ID	CITY	STATE	COUNTRY	ZIP_CODE	PHONE	MANAGER_ID
101	1	Seattle	WA	USA			6
10501	5	Bratislava		Czechozlovakia			10
201	2	Sao Paolo		Brazil			7
301	3	Lagos		Nigeria			8
401	4	Hong Kong					9

WAREHOUSE

INVENTORY

Uwaga: pokazano tylko pozycje zawierające produkty, gdzie **PRODUCT_ID** jest od 10011 do 20108

PRODUCT_ID	WAREHOUSE_ID	AMOUNT_IN_STOCK	REORDER_POINT	MAX_IN_STOCK	OUT_OF_STOCK_EXPLANATION	RESTOCK_DATE
10011	101	650	625	1100		
10012	101	600	560	1000		
10012	10501	300	300	525		
10013	101	400	400	700		
10013	10501	314	300	525		
10021	101	500	425	740		
10022	101	300	200	350		
10022	10501	502	300	525		
10023	101	400	300	525		
10023	10501	500	300	525		
20106	101	993	625	1000		
20106	201	220	150	260		
20106	10501	150	100	175		
20108	101	700	700	1225		
20108	201	166	150	260		
20108	10501	222	200	350		

PRODUCT

Uwaga: pokazano tylko produkty, gdzie **ID** jest od 10011 do 20108

ID	NAME	SHORT_DESC	LONGTEXT_ID	IMAGE_ID	SUGGESTED_WHLSL_PRICE	WHLSL_UNITS
10011	Bunny Boot	Beginner's ski boot	518	1001	150	
10012	Ace Ski Boot	Intermediate ski boot	519	1002	200	
10013	Pro Ski Boot	Advanced ski boot	520	1003	410	
10021	Bunny Ski Pole	Beginner's ski pole	528	1011	16,25	
10022	Ace Ski Pole	Intermediate ski pole	529	1012	21,95	
10023	Pro Ski Pole	Advanced ski pole	530	1013	40,95	
20106	Junior Soccer Ball	Junior soccer ball	613		11	
20108	World Cup Soccer Ball	World cup soccer ball	615		28	

ID	LAST_NAME	FIRST_NAME	USERID	START_DATE	COMMENTS	MANAGER_ID	TITLE	DEPT_ID	SALARY	COMMISSION_PCT
1	Velasquez	Carmen	cvelasqu	1990-03-03			President	50	2500	
2	Ngao	LaDoris	lngao	1990-03-08		1	VP, Operations	41	1450	
3	Nagayama	Midori	mnagayam	1991-06-17		1	VP, Sales	31	1400	
4	Quick-To-See	Mark	mquickto	1990-04-07		1	VP, Finance	10	1450	
5	Ropeburn	Audry	aropebur	1990-03-04		1	VP, Administration	50	1550	
6	Urguhart	Molly	murguhar	1991-01-18		2	Warehouse Manager	41	1200	
7	Menchu	Roberta	rmenchu	1990-05-14		2	Warehouse Manager	42	1250	
8	Biri	Ben	bbiri	1990-04-07		2	Warehouse Manager	43	1100	
9	Catchpole	Antoinette	acatchpo	1992-02-09		2	Warehouse Manager	44	1300	
10	Havel	Marta	mhavel	1991-02-27		2	Warehouse Manager	45	1307	
11	Magee	Colin	cmagee	1990-05-14		3	Sales Representative	31	1400	10
12	Giljum	Henry	hgiljum	1992-01-18		3	Sales Representative	32	1490	12,5
13	Sedeghi	Yasmin	ysedeghi	1991-02-18		3	Sales Representative	33	1515	10
14	Nguyen	Mai	mnguyen	1992-01-22		3	Sales Representative	34	1525	15
15	Dumas	Andre	adumas	1991-10-09		3	Sales Representative	35	1450	17,5
16	Maduro	Elena	emaduro	1992-02-07		6	Stock Clerk	41	1400	
17	Smith	George	gsmith	1990-03-08		6	Stock Clerk	41	940	
18	Nozaki	Akira	anozaki	1991-02-09		7	Stock Clerk	42	1200	
19	Patel	Vikram	vpatel	1991-08-06		7	Stock Clerk	42	795	
20	Newman	Chad	cnewman	1991-07-21		8	Stock Clerk	43	750	
21	Markarian	Alexander	amarkari	1991-05-26		8	Stock Clerk	43	850	
22	Chang	Eddie	echang	1990-11-30		9	Stock Clerk	44	800	
23	Patel	Radha	rpatel	1990-10-17		9	Stock Clerk	34	795	
24	Dancs	Bela	bdancs	1991-03-17		10	Stock Clerk	45	860	
25	Schwartz	Sylvie	sschwart	1991-05-09		10	Stock Clerk	45	1100	

ID	NAME
1	North America
2	South America
3	Africa / Middle East
4	Asia
5	Europe

REGION

EMP

DEPT

ID	NAME	REGION_ID
10	Finance	1
31	Sales	1
32	Sales	2
33	Sales	3
34	Sales	4
35	Sales	5
41	Operations	1
42	Operations	2
43	Operations	3
44	Operations	4
45	Operations	5
50	Administration	1

11.2. Skrypt

Poniżej zamieszczono skrypt tworzący omawiany w tym rozdziale model. Jest on automatycznie instalowany wraz z serwerem Oracle 8i.

```
Rem  Copyright (c) 1991 by Oracle Corporation
Rem  Create and populate tables and sequences to support the Summit
Rem  Sporting Goods business scenario.  These objects and data are used
Rem  in several Oracle classes and demonstration files.

set echo off

---------------------------
-- Zmiany by Artur Gramacki

-- W oryginalnej wersji rekordy wprowadzane są
-- w stronie kodowej American. Gdy ja puszczam skrypt
-- majac strone Polish, to robia sie bledy.
-- Przyklad (wysypie sie na dacie):
-- INSERT INTO ord VALUES (
--   98, 202, '31-AUG-1992', '10-SEP-1992',
--   14, 595, 'CASH', 'Y');

alter session set nls_date_language='american';
alter session set nls_date_format='dd-mon-yyyy';
alter session set nls_numeric_characters='.,';

-- Zmienilem tez w datach rok z dwucyfrowego na czterocyfrowy,
-- np. zamiast '31-AUG-92' jest teraz '31-AUG-1992'
-- Gdy bylo '31-AUG-92' to wpisywal rok 2092.

-- Zmiany by Artur Gramacki
---------------------------

Rem Create sequences.
Rem    Starting values for sequences begin at the existing maxima for
Rem    existing primary key values, plus increments.


DROP SEQUENCE customer_id;
CREATE SEQUENCE customer_id
   MINVALUE 1
   MAXVALUE 9999999
   INCREMENT BY 1
   START WITH 216
   NOCACHE
   NOORDER
   NOCYCLE;

DROP SEQUENCE dept_id;
CREATE SEQUENCE dept_id
   MINVALUE 1
   MAXVALUE 9999999
   INCREMENT BY 1
   START WITH 51
   NOCACHE
   NOORDER
   NOCYCLE;

DROP SEQUENCE emp_id;
CREATE SEQUENCE emp_id
   MINVALUE 1
   MAXVALUE 9999999
   INCREMENT BY 1
   START WITH 26
   NOCACHE
   NOORDER
   NOCYCLE;

DROP SEQUENCE image_id;
CREATE SEQUENCE image_id
   MINVALUE 1
   MAXVALUE 9999999
```

```
   INCREMENT BY 1
   START WITH 1981
   NOCACHE
   NOORDER
   NOCYCLE;

DROP SEQUENCE longtext_id;
CREATE SEQUENCE longtext_id
   MINVALUE 1
   MAXVALUE 9999999
   INCREMENT BY 1
   START WITH 1369
   NOCACHE
   NOORDER
   NOCYCLE;

DROP SEQUENCE ord_id;
CREATE SEQUENCE ord_id
   MINVALUE 1
   MAXVALUE 9999999
   INCREMENT BY 1
   START WITH 113
   NOCACHE
   NOORDER
   NOCYCLE;

DROP SEQUENCE product_id;
CREATE SEQUENCE product_id
   MINVALUE 1
   MAXVALUE 9999999
   INCREMENT BY 1
   START WITH 50537
   NOCACHE
   NOORDER
   NOCYCLE;

DROP SEQUENCE region_id;
CREATE SEQUENCE region_id
   MINVALUE 1
   MAXVALUE 9999999
   INCREMENT BY 1
   START WITH 6
   NOCACHE
   NOORDER
   NOCYCLE;

DROP SEQUENCE warehouse_id;
CREATE SEQUENCE warehouse_id
   MINVALUE 1
   MAXVALUE 9999999
   INCREMENT BY 1
   START WITH 10502
   NOCACHE
   NOORDER
   NOCYCLE;

Rem Drop tables.
DROP TABLE item;
DROP TABLE inventory;
DROP TABLE product;
DROP TABLE longtext;
DROP TABLE image;
DROP TABLE warehouse;
DROP TABLE ord;
DROP TABLE customer;
DROP TABLE emp;
DROP TABLE title;
DROP TABLE dept;
DROP TABLE region;

Rem Create and populate tables.

CREATE TABLE customer
(id                         NUMBER(7)
   CONSTRAINT customer_id_nn NOT NULL,
 name                       VARCHAR2(50)
   CONSTRAINT customer_name_nn NOT NULL,
 phone                      VARCHAR2(25),
```

```
 address                   VARCHAR2(400),
 city                      VARCHAR2(30),
 state                     VARCHAR2(20),
 country                   VARCHAR2(30),
 zip_code                  VARCHAR2(75),
 credit_rating             VARCHAR2(9),
 sales_rep_id              NUMBER(7),
 region_id                 NUMBER(7),
 comments                  VARCHAR2(255),
     CONSTRAINT customer_id_pk PRIMARY KEY (id),
     CONSTRAINT customer_credit_rating_ck
        CHECK (credit_rating IN ('EXCELLENT', 'GOOD', 'POOR')));

INSERT INTO customer VALUES (
   201, 'Unisports', '55-2066101',
   '72 Via Bahia', 'Sao Paolo', NULL, 'Brazil', NULL,
   'EXCELLENT', 12, 2, NULL);
INSERT INTO customer VALUES (
   202, 'OJ Atheletics', '81-20101',
   '6741 Takashi Blvd.', 'Osaka', NULL, 'Japan', NULL,
   'POOR', 14, 4, NULL);
INSERT INTO customer VALUES (
   203, 'Delhi Sports', '91-10351',
   '11368 Chanakya', 'New Delhi', NULL, 'India', NULL,
   'GOOD', 14, 4, NULL);
INSERT INTO customer VALUES (
   204, 'Womansport', '1-206-104-0103',
   '281 King Street', 'Seattle', 'Washington', 'USA', NULL,
   'EXCELLENT', 11, 1, NULL);
INSERT INTO customer VALUES (
   205, 'Kam''s Sporting Goods', '852-3692888',
   '15 Henessey Road', 'Hong Kong', NULL, NULL, NULL,
   'EXCELLENT', 15, 4, NULL);
INSERT INTO customer VALUES (
   206, 'Sportique', '33-2257201',
   '172 Rue de Rivoli', 'Cannes', NULL, 'France', NULL,
   'EXCELLENT', 15, 5, NULL);
INSERT INTO customer VALUES (
   207, 'Sweet Rock Sports', '234-6036201',
   '6 Saint Antoine', 'Lagos', NULL, 'Nigeria', NULL,
   'GOOD', NULL, 3, NULL);
INSERT INTO customer VALUES (
   208, 'Muench Sports', '49-527454',
   '435 Gruenestrasse', 'Stuttgart', NULL, 'Germany', NULL,
   'GOOD', 15, 5, NULL);
INSERT INTO customer VALUES (
   209, 'Beisbol Si!', '809-352689',
   '792 Playa Del Mar', 'San Pedro de Macon''s', NULL, 'Dominican Republic',
   NULL, 'EXCELLENT', 11, 1, NULL);
INSERT INTO customer VALUES (
   210, 'Futbol Sonora', '52-404562',
   '3 Via Saguaro', 'Nogales', NULL, 'Mexico', NULL,
   'EXCELLENT', 12, 2, NULL);
INSERT INTO customer VALUES (
   211, 'Kuhn''s Sports', '42-111292',
   '7 Modrany', 'Prague', NULL, 'Czechoslovakia', NULL,
   'EXCELLENT', 15, 5, NULL);
INSERT INTO customer VALUES (
   212, 'Hamada Sport', '20-1209211',
   '57A Corniche', 'Alexandria', NULL, 'Egypt', NULL,
   'EXCELLENT', 13, 3, NULL);
INSERT INTO customer VALUES (
   213, 'Big John''s Sports Emporium', '1-415-555-6281',
   '4783 18th Street', 'San Francisco', 'CA', 'USA', NULL,
   'EXCELLENT', 11, 1, NULL);
INSERT INTO customer VALUES (
   214, 'Ojibway Retail', '1-716-555-7171',
   '415 Main Street', 'Buffalo', 'NY', 'USA', NULL,
   'POOR', 11, 1, NULL);
INSERT INTO customer VALUES (
   215, 'Sporta Russia', '7-3892456',
   '6000 Yekatamina', 'Saint Petersburg', NULL, 'Russia', NULL,
   'POOR', 15, 5, NULL);
COMMIT;

CREATE TABLE dept
(id                        NUMBER(7)
   CONSTRAINT dept_id_nn NOT NULL,
```

```
 name                      VARCHAR2(25)
   CONSTRAINT dept_name_nn NOT NULL,
 region_id                 NUMBER(7),
     CONSTRAINT dept_id_pk PRIMARY KEY (id),
     CONSTRAINT dept_name_region_id_uk UNIQUE (name, region_id));

INSERT INTO dept VALUES (
   10, 'Finance', 1);
INSERT INTO dept VALUES (
   31, 'Sales', 1);
INSERT INTO dept VALUES (
   32, 'Sales', 2);
INSERT INTO dept VALUES (
   33, 'Sales', 3);
INSERT INTO dept VALUES (
   34, 'Sales', 4);
INSERT INTO dept VALUES (
   35, 'Sales', 5);
INSERT INTO dept VALUES (
   41, 'Operations', 1);
INSERT INTO dept VALUES (
   42, 'Operations', 2);
INSERT INTO dept VALUES (
   43, 'Operations', 3);
INSERT INTO dept VALUES (
   44, 'Operations', 4);
INSERT INTO dept VALUES (
   45, 'Operations', 5);
INSERT INTO dept VALUES (
   50, 'Administration', 1);
COMMIT;

CREATE TABLE emp
(id                        NUMBER(7)
   CONSTRAINT emp_id_nn NOT NULL,
 last_name                 VARCHAR2(25)
   CONSTRAINT emp_last_name_nn NOT NULL,
 first_name                VARCHAR2(25),
 userid                    VARCHAR2(8),
 start_date                DATE,
 comments                  VARCHAR2(255),
 manager_id                NUMBER(7),
 title                     VARCHAR2(25),
 dept_id                   NUMBER(7),
 salary                    NUMBER(11, 2),
 commission_pct            NUMBER(4, 2),
     CONSTRAINT emp_id_pk PRIMARY KEY (id),
     CONSTRAINT emp_userid_uk UNIQUE (userid),
     CONSTRAINT emp_commission_pct_ck
        CHECK (commission_pct IN (10, 12.5, 15, 17.5, 20)));

INSERT INTO emp VALUES (
  1, 'Velasquez', 'Carmen', 'cvelasqu',
   to_date('03-MAR-1990 8:30', 'dd-mon-yyyy hh24:mi'), NULL, NULL, 'President',
   50, 2500, NULL);
INSERT INTO emp VALUES (
   2, 'Ngao', 'LaDoris', 'lngao',
   '08-MAR-1990', NULL, 1, 'VP, Operations',
   41, 1450, NULL);
INSERT INTO emp VALUES (
   3, 'Nagayama', 'Midori', 'mnagayam',
   '17-JUN-1991', NULL, 1, 'VP, Sales',
   31, 1400, NULL);
INSERT INTO emp VALUES (
   4, 'Quick-To-See', 'Mark', 'mquickto',
   '07-APR-1990', NULL, 1, 'VP, Finance',
   10, 1450, NULL);
INSERT INTO emp VALUES (
   5, 'Ropeburn', 'Audry', 'aropebur',
   '04-MAR-1990', NULL, 1, 'VP, Administration',
   50, 1550, NULL);
INSERT INTO emp VALUES (
   6, 'Urguhart', 'Molly', 'murguhar',
   '18-JAN-1991', NULL, 2, 'Warehouse Manager',
   41, 1200, NULL);
INSERT INTO emp VALUES (
   7, 'Menchu', 'Roberta', 'rmenchu',
   '14-MAY-1990', NULL, 2, 'Warehouse Manager',
```

```
   42, 1250, NULL);
INSERT INTO emp VALUES (
   8, 'Biri', 'Ben', 'bbiri',
   '07-APR-1990', NULL, 2, 'Warehouse Manager',
   43, 1100, NULL);
INSERT INTO emp VALUES (
   9, 'Catchpole', 'Antoinette', 'acatchpo',
   '09-FEB-1992', NULL, 2, 'Warehouse Manager',
   44, 1300, NULL);
INSERT INTO emp VALUES (
   10, 'Havel', 'Marta', 'mhavel',
   '27-FEB-1991', NULL, 2, 'Warehouse Manager',
   45, 1307, NULL);
INSERT INTO emp VALUES (
   11, 'Magee', 'Colin', 'cmagee',
   '14-MAY-1990', NULL, 3, 'Sales Representative',
   31, 1400, 10);
INSERT INTO emp VALUES (
   12, 'Giljum', 'Henry', 'hgiljum',
   '18-JAN-1992', NULL, 3, 'Sales Representative',
   32, 1490, 12.5);
INSERT INTO emp VALUES (
   13, 'Sedeghi', 'Yasmin', 'ysedeghi',
   '18-FEB-1991', NULL, 3, 'Sales Representative',
   33, 1515, 10);
INSERT INTO emp VALUES (
   14, 'Nguyen', 'Mai', 'mnguyen',
   '22-JAN-1992', NULL, 3, 'Sales Representative',
   34, 1525, 15);
INSERT INTO emp VALUES (
   15, 'Dumas', 'Andre', 'adumas',
   '09-OCT-1991', NULL, 3, 'Sales Representative',
   35, 1450, 17.5);
INSERT INTO emp VALUES (
   16, 'Maduro', 'Elena', 'emaduro',
   '07-FEB-1992', NULL, 6, 'Stock Clerk',
   41, 1400, NULL);
INSERT INTO emp VALUES (
   17, 'Smith', 'George', 'gsmith',
   '08-MAR-1990', NULL, 6, 'Stock Clerk',
   41, 940, NULL);
INSERT INTO emp VALUES (
   18, 'Nozaki', 'Akira', 'anozaki',
   '09-FEB-1991', NULL, 7, 'Stock Clerk',
   42, 1200, NULL);
INSERT INTO emp VALUES (
   19, 'Patel', 'Vikram', 'vpatel',
   '06-AUG-1991', NULL, 7, 'Stock Clerk',
   42, 795, NULL);
INSERT INTO emp VALUES (
   20, 'Newman', 'Chad', 'cnewman',
   '21-JUL-1991', NULL, 8, 'Stock Clerk',
   43, 750, NULL);
INSERT INTO emp VALUES (
   21, 'Markarian', 'Alexander', 'amarkari',
   '26-MAY-1991', NULL, 8, 'Stock Clerk',
   43, 850, NULL);
INSERT INTO emp VALUES (
   22, 'Chang', 'Eddie', 'echang',
   '30-NOV-1990', NULL, 9, 'Stock Clerk',
   44, 800, NULL);
INSERT INTO emp VALUES (
   23, 'Patel', 'Radha', 'rpatel',
   '17-OCT-1990', NULL, 9, 'Stock Clerk',
   34, 795, NULL);
INSERT INTO emp VALUES (
   24, 'Dancs', 'Bela', 'bdancs',
   '17-MAR-1991', NULL, 10, 'Stock Clerk',
   45, 860, NULL);
INSERT INTO emp VALUES (
   25, 'Schwartz', 'Sylvie', 'sschwart',
   '09-MAY-1991', NULL, 10, 'Stock Clerk',
   45, 1100, NULL);
COMMIT;

CREATE TABLE image
(id                          NUMBER(7)
   CONSTRAINT image_id_nn NOT NULL,
```

```
 format                    VARCHAR2(25),
 use_filename              VARCHAR2(1),
 filename                  VARCHAR2(255),
 image                     LONG RAW,
     CONSTRAINT image_id_pk
        PRIMARY KEY (id),
     CONSTRAINT image_format_ck
        CHECK (format in ('JFIFF', 'JTIFF')),
     CONSTRAINT image_use_filename_ck
        CHECK (use_filename in ('Y', 'N')));

INSERT INTO image VALUES (
   1001, 'JTIFF', 'Y', 'bunboot.tif', NULL);
INSERT INTO image VALUES (
   1002, 'JTIFF', 'Y', 'aceboot.tif', NULL);
INSERT INTO image VALUES (
   1003, 'JTIFF', 'Y', 'proboot.tif', NULL);
INSERT INTO image VALUES (
   1011, 'JTIFF', 'Y', 'bunpole.tif', NULL);
INSERT INTO image VALUES (
   1012, 'JTIFF', 'Y', 'acepole.tif', NULL);
INSERT INTO image VALUES (
   1013, 'JTIFF', 'Y', 'propole.tif', NULL);
INSERT INTO image VALUES (
   1291, 'JTIFF', 'Y', 'gpbike.tif', NULL);
INSERT INTO image VALUES (
   1296, 'JTIFF', 'Y', 'himbike.tif', NULL);
INSERT INTO image VALUES (
   1829, 'JTIFF', 'Y', 'safthelm.tif', NULL);
INSERT INTO image VALUES (
   1381, 'JTIFF', 'Y', 'probar.tif', NULL);
INSERT INTO image VALUES (
   1382, 'JTIFF', 'Y', 'curlbar.tif', NULL);
INSERT INTO image VALUES (
   1119, 'JTIFF', 'Y', 'baseball.tif', NULL);
INSERT INTO image VALUES (
   1223, 'JTIFF', 'Y', 'chaphelm.tif', NULL);
INSERT INTO image VALUES (
   1367, 'JTIFF', 'Y', 'grglove.tif', NULL);
INSERT INTO image VALUES (
   1368, 'JTIFF', 'Y', 'alglove.tif', NULL);
INSERT INTO image VALUES (
   1369, 'JTIFF', 'Y', 'stglove.tif', NULL);
INSERT INTO image VALUES (
   1480, 'JTIFF', 'Y', 'cabbat.tif', NULL);
INSERT INTO image VALUES (
   1482, 'JTIFF', 'Y', 'pucbat.tif', NULL);
INSERT INTO image VALUES (
   1486, 'JTIFF', 'Y', 'winbat.tif', NULL);
COMMIT;

CREATE TABLE inventory
(product_id                NUMBER(7)
   CONSTRAINT inventory_product_id_nn NOT NULL,
 warehouse_id              NUMBER(7)
   CONSTRAINT inventory_warehouse_id_nn NOT NULL,
 amount_in_stock           NUMBER(9),
 reorder_point             NUMBER(9),
 max_in_stock              NUMBER(9),
 out_of_stock_explanation  VARCHAR2(255),
 restock_date              DATE,
     CONSTRAINT inventory_prodid_warid_pk
        PRIMARY KEY (product_id, warehouse_id));

INSERT INTO inventory VALUES (
   10011, 101, 650, 625, 1100, NULL, NULL);
INSERT INTO inventory VALUES (
   10012, 101, 600, 560, 1000, NULL, NULL);
INSERT INTO inventory VALUES (
   10013, 101, 400, 400, 700, NULL, NULL);
INSERT INTO inventory VALUES (
   10021, 101, 500, 425, 740, NULL, NULL);
INSERT INTO inventory VALUES (
   10022, 101, 300, 200, 350, NULL, NULL);
INSERT INTO inventory VALUES (
   10023, 101, 400, 300, 525, NULL, NULL);
INSERT INTO inventory VALUES (
   20106, 101, 993, 625, 1000, NULL, NULL);
```

```
INSERT INTO inventory VALUES (
   20108, 101, 700, 700, 1225, NULL, NULL);
INSERT INTO inventory VALUES (
   20201, 101, 802, 800, 1400, NULL, NULL);
INSERT INTO inventory VALUES (
   20510, 101, 1389, 850, 1400, NULL, NULL);
INSERT INTO inventory VALUES (
   20512, 101, 850, 850, 1450, NULL, NULL);
INSERT INTO inventory VALUES (
   30321, 101, 2000, 1500, 2500, NULL, NULL);
INSERT INTO inventory VALUES (
   30326, 101, 2100, 2000, 3500, NULL, NULL);
INSERT INTO inventory VALUES (
   30421, 101, 1822, 1800, 3150, NULL, NULL);
INSERT INTO inventory VALUES (
   30426, 101, 2250, 2000, 3500, NULL, NULL);
INSERT INTO inventory VALUES (
   30433, 101, 650, 600, 1050, NULL, NULL);
INSERT INTO inventory VALUES (
   32779, 101, 2120, 1250, 2200, NULL, NULL);
INSERT INTO inventory VALUES (
   32861, 101, 505, 500, 875, NULL, NULL);
INSERT INTO inventory VALUES (
   40421, 101, 578, 350, 600, NULL, NULL);
INSERT INTO inventory VALUES (
   40422, 101, 0, 350, 600, 'Phenomenal sales...', '08-FEB-1993');
INSERT INTO inventory VALUES (
   41010, 101, 250, 250, 437, NULL, NULL);
INSERT INTO inventory VALUES (
   41020, 101, 471, 450, 750, NULL, NULL);
INSERT INTO inventory VALUES (
   41050, 101, 501, 450, 750, NULL, NULL);
INSERT INTO inventory VALUES (
   41080, 101, 400, 400, 700, NULL, NULL);
INSERT INTO inventory VALUES (
   41100, 101, 350, 350, 600, NULL, NULL);
INSERT INTO inventory VALUES (
   50169, 101, 2530, 1500, 2600, NULL, NULL);
INSERT INTO inventory VALUES (
   50273, 101, 233, 200, 350, NULL, NULL);
INSERT INTO inventory VALUES (
   50417, 101, 518, 500, 875, NULL, NULL);
INSERT INTO inventory VALUES (
   50418, 101, 244, 100, 275, NULL, NULL);
INSERT INTO inventory VALUES (
   50419, 101, 230, 120, 310, NULL, NULL);
INSERT INTO inventory VALUES (
   50530, 101, 669, 400, 700, NULL, NULL);
INSERT INTO inventory VALUES (
   50532, 101, 0, 100, 175, 'Wait for Spring.', '12-APR-1993');
INSERT INTO inventory VALUES (
   50536, 101, 173, 100, 175, NULL, NULL);
INSERT INTO inventory VALUES (
   20106, 201, 220, 150, 260, NULL, NULL);
INSERT INTO inventory VALUES (
   20108, 201, 166, 150, 260, NULL, NULL);
INSERT INTO inventory VALUES (
   20201, 201, 320, 200, 350, NULL, NULL);
INSERT INTO inventory VALUES (
   20510, 201, 175, 100, 175, NULL, NULL);
INSERT INTO inventory VALUES (
   20512, 201, 162, 100, 175, NULL, NULL);
INSERT INTO inventory VALUES (
   30321, 201, 96, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   30326, 201, 147, 120, 210, NULL, NULL);
INSERT INTO inventory VALUES (
   30421, 201, 102, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   30426, 201, 200, 120, 210, NULL, NULL);
INSERT INTO inventory VALUES (
   30433, 201, 130, 130, 230, NULL, NULL);
INSERT INTO inventory VALUES (
   32779, 201, 180, 150, 260, NULL, NULL);
INSERT INTO inventory VALUES (
   32861, 201, 132, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   50169, 201, 225, 220, 385, NULL, NULL);
```

```
INSERT INTO inventory VALUES (
   50273, 201, 75, 60, 100, NULL, NULL);
INSERT INTO inventory VALUES (
   50417, 201, 82, 60, 100, NULL, NULL);
INSERT INTO inventory VALUES (
   50418, 201, 98, 60, 100, NULL, NULL);
INSERT INTO inventory VALUES (
   50419, 201, 77, 60, 100, NULL, NULL);
INSERT INTO inventory VALUES (
   50530, 201, 62, 60, 100, NULL, NULL);
INSERT INTO inventory VALUES (
   50532, 201, 67, 60, 100, NULL, NULL);
INSERT INTO inventory VALUES (
   50536, 201, 97, 60, 100, NULL, NULL);
INSERT INTO inventory VALUES (
   20510, 301, 69, 40, 100, NULL, NULL);
INSERT INTO inventory VALUES (
   20512, 301, 28, 20, 50, NULL, NULL);
INSERT INTO inventory VALUES (
   30321, 301, 85, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   30421, 301, 102, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   30433, 301, 35, 20, 35, NULL, NULL);
INSERT INTO inventory VALUES (
   32779, 301, 102, 95, 175, NULL, NULL);
INSERT INTO inventory VALUES (
   32861, 301, 57, 50, 100, NULL, NULL);
INSERT INTO inventory VALUES (
   40421, 301, 70, 40, 70, NULL, NULL);
INSERT INTO inventory VALUES (
   40422, 301, 65, 40, 70, NULL, NULL);
INSERT INTO inventory VALUES (
   41010, 301, 59, 40, 70, NULL, NULL);
INSERT INTO inventory VALUES (
   41020, 301, 61, 40, 70, NULL, NULL);
INSERT INTO inventory VALUES (
   41050, 301, 49, 40, 70, NULL, NULL);
INSERT INTO inventory VALUES (
   41080, 301, 50, 40, 70, NULL, NULL);
INSERT INTO inventory VALUES (
   41100, 301, 42, 40, 70, NULL, NULL);
INSERT INTO inventory VALUES (
   20510, 401, 88, 50, 100, NULL, NULL);
INSERT INTO inventory VALUES (
   20512, 401, 75, 75, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   30321, 401, 102, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   30326, 401, 113, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   30421, 401, 85, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   30426, 401, 135, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   30433, 401, 0, 100, 175, 'A defective shipment was sent to Hong Kong ' ||
   'and needed to be returned. The soonest ACME can turn this around is ' ||
   'early February.', '07-SEP-1992');
INSERT INTO inventory VALUES (
   32779, 401, 135, 100, 175, NULL, NULL);
INSERT INTO inventory VALUES (
   32861, 401, 250, 150, 250, NULL, NULL);
INSERT INTO inventory VALUES (
   40421, 401, 47, 40, 70, NULL, NULL);
INSERT INTO inventory VALUES (
   40422, 401, 50, 40, 70, NULL, NULL);
INSERT INTO inventory VALUES (
   41010, 401, 80, 70, 220, NULL, NULL);
INSERT INTO inventory VALUES (
   41020, 401, 91, 70, 220, NULL, NULL);
INSERT INTO inventory VALUES (
   41050, 401, 169, 70, 220, NULL, NULL);
INSERT INTO inventory VALUES (
   41080, 401, 100, 70, 220, NULL, NULL);
INSERT INTO inventory VALUES (
   41100, 401, 75, 70, 220, NULL, NULL);
INSERT INTO inventory VALUES (
   50169, 401, 240, 200, 350, NULL, NULL);
```

```
INSERT INTO inventory VALUES (
   50273, 401, 224, 150, 280, NULL, NULL);
INSERT INTO inventory VALUES (
   50417, 401, 130, 120, 210, NULL, NULL);
INSERT INTO inventory VALUES (
   50418, 401, 156, 100, 175, NULL, NULL);
INSERT INTO inventory VALUES (
   50419, 401, 151, 150, 280, NULL, NULL);
INSERT INTO inventory VALUES (
   50530, 401, 119, 100, 175, NULL, NULL);
INSERT INTO inventory VALUES (
   50532, 401, 233, 200, 350, NULL, NULL);
INSERT INTO inventory VALUES (
   50536, 401, 138, 100, 175, NULL, NULL);
INSERT INTO inventory VALUES (
   10012, 10501, 300, 300, 525, NULL, NULL);
INSERT INTO inventory VALUES (
   10013, 10501, 314, 300, 525, NULL, NULL);
INSERT INTO inventory VALUES (
   10022, 10501, 502, 300, 525, NULL, NULL);
INSERT INTO inventory VALUES (
   10023, 10501, 500, 300, 525, NULL, NULL);
INSERT INTO inventory VALUES (
   20106, 10501, 150, 100, 175, NULL, NULL);
INSERT INTO inventory VALUES (
   20108, 10501, 222, 200, 350, NULL, NULL);
INSERT INTO inventory VALUES (
   20201, 10501, 275, 200, 350, NULL, NULL);
INSERT INTO inventory VALUES (
   20510, 10501, 57, 50, 87, NULL, NULL);
INSERT INTO inventory VALUES (
   20512, 10501, 62, 50, 87, NULL, NULL);
INSERT INTO inventory VALUES (
   30321, 10501, 194, 150, 275, NULL, NULL);
INSERT INTO inventory VALUES (
   30326, 10501, 277, 250, 440, NULL, NULL);
INSERT INTO inventory VALUES (
   30421, 10501, 190, 150, 275, NULL, NULL);
INSERT INTO inventory VALUES (
   30426, 10501, 423, 250, 450, NULL, NULL);
INSERT INTO inventory VALUES (
   30433, 10501, 273, 200, 350, NULL, NULL);
INSERT INTO inventory VALUES (
   32779, 10501, 280, 200, 350, NULL, NULL);
INSERT INTO inventory VALUES (
   32861, 10501, 288, 200, 350, NULL, NULL);
INSERT INTO inventory VALUES (
   40421, 10501, 97, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   40422, 10501, 90, 80, 140, NULL, NULL);
INSERT INTO inventory VALUES (
   41010, 10501, 151, 140, 245, NULL, NULL);
INSERT INTO inventory VALUES (
   41020, 10501, 224, 140, 245, NULL, NULL);
INSERT INTO inventory VALUES (
   41050, 10501, 157, 140, 245, NULL, NULL);
INSERT INTO inventory VALUES (
   41080, 10501, 159, 140, 245, NULL, NULL);
INSERT INTO inventory VALUES (
   41100, 10501, 141, 140, 245, NULL, NULL);
COMMIT;


CREATE TABLE item
(ord_id                     NUMBER(7)
   CONSTRAINT item_ord_id_nn NOT NULL,
 item_id                    NUMBER(7)
   CONSTRAINT item_item_id_nn NOT NULL,
 product_id                 NUMBER(7)
   CONSTRAINT item_product_id_nn NOT NULL,
 price                      NUMBER(11, 2),
 quantity                   NUMBER(9),
 quantity_shipped           NUMBER(9),
     CONSTRAINT item_ordid_itemid_pk PRIMARY KEY (ord_id, item_id),
     CONSTRAINT item_ordid_prodid_uk UNIQUE (ord_id, product_id));

INSERT INTO item VALUES (
   100, 1, 10011, 135, 500, 500);
```

```
INSERT INTO item VALUES (
   100, 2, 10013, 380, 400, 400);
INSERT INTO item VALUES (
   100, 3, 10021, 14, 500, 500);
INSERT INTO item VALUES (
   100, 5, 30326, 582, 600, 600);
INSERT INTO item VALUES (
   100, 7, 41010, 8, 250, 250);
INSERT INTO item VALUES (
   100, 6, 30433, 20, 450, 450);
INSERT INTO item VALUES (
   100, 4, 10023, 36, 400, 400);
INSERT INTO item VALUES (
   101, 1, 30421, 16, 15, 15);
INSERT INTO item VALUES (
   101, 3, 41010, 8, 20, 20);
INSERT INTO item VALUES (
   101, 5, 50169, 4.29, 40, 40);
INSERT INTO item VALUES (
   101, 6, 50417, 80, 27, 27);
INSERT INTO item VALUES (
   101, 7, 50530, 45, 50, 50);
INSERT INTO item VALUES (
   101, 4, 41100, 45, 35, 35);
INSERT INTO item VALUES (
   101, 2, 40422, 50, 30, 30);
INSERT INTO item VALUES (
   102, 1, 20108, 28, 100, 100);
INSERT INTO item VALUES (
   102, 2, 20201, 123, 45, 45);
INSERT INTO item VALUES (
   103, 1, 30433, 20, 15, 15);
INSERT INTO item VALUES (
   103, 2, 32779, 7, 11, 11);
INSERT INTO item VALUES (
   104, 1, 20510, 9, 7, 7);
INSERT INTO item VALUES (
   104, 4, 30421, 16, 35, 35);
INSERT INTO item VALUES (
   104, 2, 20512, 8, 12, 12);
INSERT INTO item VALUES (
   104, 3, 30321, 1669, 19, 19);
INSERT INTO item VALUES (
   105, 1, 50273, 22.89, 16, 16);
INSERT INTO item VALUES (
   105, 3, 50532, 47, 28, 28);
INSERT INTO item VALUES (
   105, 2, 50419, 80, 13, 13);
INSERT INTO item VALUES (
   106, 1, 20108, 28, 46, 46);
INSERT INTO item VALUES (
   106, 4, 50273, 22.89, 75, 75);
INSERT INTO item VALUES (
   106, 5, 50418, 75, 98, 98);
INSERT INTO item VALUES (
   106, 6, 50419, 80, 27, 27);
INSERT INTO item VALUES (
   106, 2, 20201, 123, 21, 21);
INSERT INTO item VALUES (
   106, 3, 50169, 4.29, 125, 125);
INSERT INTO item VALUES (
   107, 1, 20106, 11, 50, 50);
INSERT INTO item VALUES (
   107, 3, 20201, 115, 130, 130);
INSERT INTO item VALUES (
   107, 5, 30421, 16, 55, 55);
INSERT INTO item VALUES (
   107, 4, 30321, 1669, 75, 75);
INSERT INTO item VALUES (
   107, 2, 20108, 28, 22, 22);
INSERT INTO item VALUES (
   108, 1, 20510, 9, 9, 9);
INSERT INTO item VALUES (
   108, 6, 41080, 35, 50, 50);
INSERT INTO item VALUES (
   108, 7, 41100, 45, 42, 42);
INSERT INTO item VALUES (
   108, 5, 32861, 60, 57, 57);
```

```
INSERT INTO item VALUES (
   108, 2, 20512, 8, 18, 18);
INSERT INTO item VALUES (
   108, 4, 32779, 7, 60, 60);
INSERT INTO item VALUES (
   108, 3, 30321, 1669, 85, 85);
INSERT INTO item VALUES (
   109, 1, 10011, 140, 150, 150);
INSERT INTO item VALUES (
   109, 5, 30426, 18.25, 500, 500);
INSERT INTO item VALUES (
   109, 7, 50418, 75, 43, 43);
INSERT INTO item VALUES (
   109, 6, 32861, 60, 50, 50);
INSERT INTO item VALUES (
   109, 4, 30326, 582, 1500, 1500);
INSERT INTO item VALUES (
   109, 2, 10012, 175, 600, 600);
INSERT INTO item VALUES (
   109, 3, 10022, 21.95, 300, 300);
INSERT INTO item VALUES (
   110, 1, 50273, 22.89, 17, 17);
INSERT INTO item VALUES (
   110, 2, 50536, 50, 23, 23);
INSERT INTO item VALUES (
   111, 1, 40421, 65, 27, 27);
INSERT INTO item VALUES (
   111, 2, 41080, 35, 29, 29);
INSERT INTO item VALUES (
   97, 1, 20106, 9, 1000, 1000);
INSERT INTO item VALUES (
   97, 2, 30321, 1500, 50, 50);
INSERT INTO item VALUES (
   98, 1, 40421, 85, 7, 7);
INSERT INTO item VALUES (
   99, 1, 20510, 9, 18, 18);
INSERT INTO item VALUES (
   99, 2, 20512, 8, 25, 25);
INSERT INTO item VALUES (
   99, 3, 50417, 80, 53, 53);
INSERT INTO item VALUES (
   99, 4, 50530, 45, 69, 69);
INSERT INTO item VALUES (
   112, 1, 20106, 11, 50, 50);
COMMIT;


CREATE TABLE longtext
(id                          NUMBER(7)
   CONSTRAINT longtext_id_nn NOT NULL,
 use_filename                VARCHAR2(1),
 filename                    VARCHAR2(255),
 text                        VARCHAR2(2000),
     CONSTRAINT longtext_id_pk PRIMARY KEY (id),
     CONSTRAINT longtext_use_filename_ck
        CHECK (use_filename in ('Y', 'N')));

INSERT INTO longtext VALUES (
   1017, 'N', NULL,
   'Protective knee pads for any number of physical activities including ' ||
   'bicycling and skating (4-wheel, in-line, and ice).  Also provide ' ||
   'support for stress activities such as weight-lifting.  Velcro belts ' ||
   'allow easy adjustment for any size and snugness of fit.  Hardened ' ||
   'plastic shell comes in a variety of colors, so you can buy a pair to ' ||
   'match every outfit.  Can also be worn at the beach to cover ' ||
   'particularly ugly knees.');
INSERT INTO longtext VALUES (
   1019, 'N', NULL,
   'Protective elbow pads for any number of physical activities including ' ||
   'bicycling and skating (4-wheel, in-line, and ice).  Also provide ' ||
   'support for stress activities such as weight-lifting.  Velcro belts ' ||
   'allow easy adjustment for any size and snugness of fit.  Hardened ' ||
   'plastic shell comes in a variety of colors, so you can buy a pair to ' ||
   'match every outfit.');
INSERT INTO longtext VALUES (
   1037, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   1039, NULL, NULL, NULL);
```

```
INSERT INTO longtext VALUES (
   1043, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   1286, 'N', NULL,
   'Don''t slack off--try the Slaker Water Bottle.  With its 1 quart ' ||
   'capacity, this is the only water bottle you''ll need.  It''s ' ||
   'lightweight, durable, and guaranteed for life to be leak proof.  It ' ||
   'comes with a convenient velcro strap so it ' ||
   'can be conveniently attached to your bike or other sports equipment.');
INSERT INTO longtext VALUES (
   1368, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   517, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   518, 'N', NULL,
   'Perfect for the beginner.  Rear entry (easy to put on with only one ' ||
   'buckle), weight control adjustment on side of boot for easy access, ' ||
   'comes in a wide variety of colors to match every outfit.');
INSERT INTO longtext VALUES (
   519, 'N', NULL,
   'If you have mastered the basic techniques you are ready for the Ace Ski ' ||
   'Boot.  This intermediate boot comes as a package with self adjustable ' ||
   'bindings that will adapt to your skill and speed. The boot is designed ' ||
   'for extra grip on slopes and jumps.');
INSERT INTO longtext VALUES (
   520, 'N', NULL,
   'The Pro ski boot is an advanced boot that combines high tech and ' ||
   'comfort.  It''s made of fibre that will mould to your foot with body ' ||
   'heat.  If you''re after perfection, don''t look any further: this is it!');
INSERT INTO longtext VALUES (
   527, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   528, 'N', NULL,
   'Lightweight aluminum pole, comes in a variety of sizes and neon ' ||
   'colors.  Comfortable adjustable straps.');
INSERT INTO longtext VALUES (
   529, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   530, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   557, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   587, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   607, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   613, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   615, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   676, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   708, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   780, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   828, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   833, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   924, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   925, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   926, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   927, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   928, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   929, NULL, NULL, NULL);
INSERT INTO longtext VALUES (
   933, 'N', NULL,
   'The widest, strongest, and knobbiest tires for mountain bike ' ||
   'enthusiasts.  Guaranteed to withstand pummelling that will reduce most ' ||
   'bicycles (except for the Himalayan) to scrap iron.  These tires can ' ||
   'carry you to places where nobody would want to bicycle.  Sizes to ' ||
   'fit all makes of mountain bike including wide and super wide rims.  ' ||
```

```
   'Steel-banded radial models are also available by direct factory order.');
INSERT INTO longtext VALUES (
   940, NULL, NULL, NULL);
COMMIT;


CREATE TABLE ord
(id                          NUMBER(7)
   CONSTRAINT ord_id_nn NOT NULL,
 customer_id                 NUMBER(7)
   CONSTRAINT ord_customer_id_nn NOT NULL,
 date_ordered                DATE,
 date_shipped                DATE,
 sales_rep_id                NUMBER(7),
 total                       NUMBER(11, 2),
 payment_type                VARCHAR2(6),
 order_filled                VARCHAR2(1),
     CONSTRAINT ord_id_pk PRIMARY KEY (id),
     CONSTRAINT ord_payment_type_ck
        CHECK (payment_type in ('CASH', 'CREDIT')),
     CONSTRAINT ord_order_filled_ck
        CHECK (order_filled in ('Y', 'N')));

INSERT INTO ord VALUES (
   100, 204, '31-AUG-1992', '10-SEP-1992',
   11, 601100, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   101, 205, '31-AUG-1992', '15-SEP-1992',
   14, 8056.6, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   102, 206, '01-SEP-1992', '08-SEP-1992',
   15, 8335, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   103, 208, '02-SEP-1992', '22-SEP-1992',
   15, 377, 'CASH', 'Y');
INSERT INTO ord VALUES (
   104, 208, '03-SEP-1992', '23-SEP-1992',
   15, 32430, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   105, 209, '04-SEP-1992', '18-SEP-1992',
   11, 2722.24, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   106, 210, '07-SEP-1992', '15-SEP-1992',
   12, 15634, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   107, 211, '07-SEP-1992', '21-SEP-1992',
   15, 142171, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   108, 212, '07-SEP-1992', '10-SEP-1992',
   13, 149570, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   109, 213, '08-SEP-1992', '28-SEP-1992',
   11, 1020935, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   110, 214, '09-SEP-1992', '21-SEP-1992',
   11, 1539.13, 'CASH', 'Y');
INSERT INTO ord VALUES (
   111, 204, '09-SEP-1992', '21-SEP-1992',
   11, 2770, 'CASH', 'Y');
INSERT INTO ord VALUES (
   97, 201, '28-AUG-1992', '17-SEP-1992',
   12, 84000, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   98, 202, '31-AUG-1992', '10-SEP-1992',
   14, 595, 'CASH', 'Y');
INSERT INTO ord VALUES (
   99, 203, '31-AUG-1992', '18-SEP-1992',
   14, 7707, 'CREDIT', 'Y');
INSERT INTO ord VALUES (
   112, 210, '31-AUG-1992', '10-SEP-1992',
   12, 550, 'CREDIT', 'Y');
COMMIT;


CREATE TABLE product
(id                          NUMBER(7)
   CONSTRAINT product_id_nn NOT NULL,
 name                        VARCHAR2(50)
```

```
   CONSTRAINT product_name_nn NOT NULL,
 short_desc                 VARCHAR2(255),
 longtext_id                NUMBER(7),
 image_id                   NUMBER(7),
 suggested_whlsl_price      NUMBER(11, 2),
 whlsl_units                VARCHAR2(25),
     CONSTRAINT product_id_pk PRIMARY KEY (id),
     CONSTRAINT product_name_uk UNIQUE (name));

INSERT INTO product VALUES (
   10011, 'Bunny Boot',
   'Beginner''s ski boot',
   518, 1001,
   150, NULL);
INSERT INTO product VALUES (
   10012, 'Ace Ski Boot',
   'Intermediate ski boot',
   519, 1002,
   200, NULL);
INSERT INTO product VALUES (
   10013, 'Pro Ski Boot',
   'Advanced ski boot',
   520, 1003,
   410, NULL);
INSERT INTO product VALUES (
   10021, 'Bunny Ski Pole',
   'Beginner''s ski pole',
   528, 1011,
   16.25, NULL);
INSERT INTO product VALUES (
   10022, 'Ace Ski Pole',
   'Intermediate ski pole',
   529, 1012,
   21.95, NULL);
INSERT INTO product VALUES (
   10023, 'Pro Ski Pole',
   'Advanced ski pole',
   530, 1013,
   40.95, NULL);
INSERT INTO product VALUES (
   20106, 'Junior Soccer Ball',
   'Junior soccer ball',
   613, NULL,
   11, NULL);
INSERT INTO product VALUES (
   20108, 'World Cup Soccer Ball',
   'World cup soccer ball',
   615, NULL,
   28, NULL);
INSERT INTO product VALUES (
   20201, 'World Cup Net',
   'World cup net',
   708, NULL,
   123, NULL);
INSERT INTO product VALUES (
   20510, 'Black Hawk Knee Pads',
   'Knee pads, pair',
   1017, NULL,
   9, NULL);
INSERT INTO product VALUES (
   20512, 'Black Hawk Elbow Pads',
   'Elbow pads, pair',
   1019, NULL,
   8, NULL);
INSERT INTO product VALUES (
   30321, 'Grand Prix Bicycle',
   'Road bicycle',
   828, 1291,
   1669, NULL);
INSERT INTO product VALUES (
   30326, 'Himalaya Bicycle',
   'Mountain bicycle',
   833, 1296,
   582, NULL);
INSERT INTO product VALUES (
   30421, 'Grand Prix Bicycle Tires',
   'Road bicycle tires',
   927, NULL,
```

```
   16, NULL);
INSERT INTO product VALUES (
   30426, 'Himalaya Tires',
   'Mountain bicycle tires',
   933, NULL,
   18.25, NULL);
INSERT INTO product VALUES (
   30433, 'New Air Pump',
   'Tire pump',
   940, NULL,
   20, NULL);
INSERT INTO product VALUES (
   32779, 'Slaker Water Bottle',
   'Water bottle',
   1286, NULL,
   7, NULL);
INSERT INTO product VALUES (
   32861, 'Safe-T Helmet',
   'Bicycle helmet',
   1368, 1829,
   60, NULL);
INSERT INTO product VALUES (
   40421, 'Alexeyer Pro Lifting Bar',
   'Straight bar',
   928, 1381,
   65, NULL);
INSERT INTO product VALUES (
   40422, 'Pro Curling Bar',
   'Curling bar',
   929, 1382,
   50, NULL);
INSERT INTO product VALUES (
   41010, 'Prostar 10 Pound Weight',
   'Ten pound weight',
   517, NULL,
   8, NULL);
INSERT INTO product VALUES (
   41020, 'Prostar 20 Pound Weight',
   'Twenty pound weight',
   527, NULL,
   12, NULL);
INSERT INTO product VALUES (
   41050, 'Prostar 50 Pound Weight',
   'Fifty pound weight',
   557, NULL,
   25, NULL);
INSERT INTO product VALUES (
   41080, 'Prostar 80 Pound Weight',
   'Eighty pound weight',
   587, NULL,
   35, NULL);
INSERT INTO product VALUES (
   41100, 'Prostar 100 Pound Weight',
   'One hundred pound weight',
   607, NULL,
   45, NULL);
INSERT INTO product VALUES (
   50169, 'Major League Baseball',
   'Baseball',
   676, 1119,
   4.29, NULL);
INSERT INTO product VALUES (
   50273, 'Chapman Helmet',
   'Batting helmet',
   780, 1223,
   22.89, NULL);
INSERT INTO product VALUES (
   50417, 'Griffey Glove',
   'Outfielder''s glove',
   924, 1367,
   80, NULL);
INSERT INTO product VALUES (
   50418, 'Alomar Glove',
   'Infielder''s glove',
   925, 1368,
   75, NULL);
INSERT INTO product VALUES (
   50419, 'Steinbach Glove',
```

```
   'Catcher''s glove',
   926, 1369,
   80, NULL);
INSERT INTO product VALUES (
   50530, 'Cabrera Bat',
   'Thirty inch bat',
   1037, 1480,
   45, NULL);
INSERT INTO product VALUES (
   50532, 'Puckett Bat',
   'Thirty-two inch bat',
   1039, 1482,
   47, NULL);
INSERT INTO product VALUES (
   50536, 'Winfield Bat',
   'Thirty-six inch bat',
   1043, 1486,
   50, NULL);
COMMIT;


CREATE TABLE region
(id                          NUMBER(7)
   CONSTRAINT region_id_nn NOT NULL,
 name                        VARCHAR2(50)
   CONSTRAINT region_name_nn NOT NULL,
     CONSTRAINT region_id_pk PRIMARY KEY (id),
     CONSTRAINT region_name_uk UNIQUE (name));

INSERT INTO region VALUES (
   1, 'North America');
INSERT INTO region VALUES (
   2, 'South America');
INSERT INTO region VALUES (
   3, 'Africa / Middle East');
INSERT INTO region VALUES (
   4, 'Asia');
INSERT INTO region VALUES (
   5, 'Europe');
COMMIT;


CREATE TABLE title
(title                       VARCHAR2(25)
   CONSTRAINT title_title_nn NOT NULL,
       CONSTRAINT title_title_pk PRIMARY KEY (title));

INSERT INTO title VALUES ('President');
INSERT INTO title VALUES ('Sales Representative');
INSERT INTO title VALUES ('Stock Clerk');
INSERT INTO title VALUES ('VP, Administration');
INSERT INTO title VALUES ('VP, Finance');
INSERT INTO title VALUES ('VP, Operations');
INSERT INTO title VALUES ('VP, Sales');
INSERT INTO title VALUES ('Warehouse Manager');
COMMIT;


CREATE TABLE warehouse
(id                          NUMBER(7)
   CONSTRAINT warehouse_id_nn NOT NULL,
 region_id                   NUMBER(7)
   CONSTRAINT warehouse_region_id_nn NOT NULL,
 address                     LONG,
 city                        VARCHAR2(30),
 state                       VARCHAR2(20),
 country                     VARCHAR2(30),
 zip_code                    VARCHAR2(75),
 phone                       VARCHAR2(25),
 manager_id                  NUMBER(7),
     CONSTRAINT warehouse_id_pk PRIMARY KEY (id));

INSERT INTO warehouse VALUES (
   101, 1,
   '283 King Street',
   'Seattle', 'WA', 'USA',
   NULL,
   NULL, 6);
```

```
INSERT INTO warehouse VALUES (
   10501, 5,
   '5 Modrany',
   'Bratislava', NULL, 'Czechozlovakia',
   NULL,
   NULL, 10);
INSERT INTO warehouse VALUES (
   201, 2,
   '68 Via Centrale',
   'Sao Paolo', NULL, 'Brazil',
   NULL,
   NULL, 7);
INSERT INTO warehouse VALUES (
   301, 3,
   '6921 King Way',
   'Lagos', NULL, 'Nigeria',
   NULL,
   NULL, 8);
INSERT INTO warehouse VALUES (
   401, 4,
   '86 Chu Street',
   'Hong Kong', NULL, NULL,
   NULL,
   NULL, 9);
COMMIT;

Rem Add foreign key constraints.

ALTER TABLE dept
   ADD CONSTRAINT dept_region_id_fk
   FOREIGN KEY (region_id) REFERENCES region (id);
ALTER TABLE emp
   ADD CONSTRAINT emp_manager_id_fk
   FOREIGN KEY (manager_id) REFERENCES emp (id);
ALTER TABLE emp
   ADD CONSTRAINT emp_dept_id_fk
   FOREIGN KEY (dept_id) REFERENCES dept (id);
ALTER TABLE emp
   ADD CONSTRAINT emp_title_fk
   FOREIGN KEY (title) REFERENCES title (title);
ALTER TABLE customer
   ADD CONSTRAINT sales_rep_id_fk
   FOREIGN KEY (sales_rep_id) REFERENCES emp (id);
ALTER TABLE customer
   ADD CONSTRAINT customer_region_id_fk
   FOREIGN KEY (region_id) REFERENCES region (id);
ALTER TABLE ord
   ADD CONSTRAINT ord_customer_id_fk
   FOREIGN KEY (customer_id) REFERENCES customer (id);
ALTER TABLE ord
   ADD CONSTRAINT ord_sales_rep_id_fk
   FOREIGN KEY (sales_rep_id) REFERENCES emp (id);
ALTER TABLE product
   ADD CONSTRAINT product_image_id_fk
   FOREIGN KEY (image_id) REFERENCES image (id);
ALTER TABLE product
   ADD CONSTRAINT product_longtext_id_fk
   FOREIGN KEY (longtext_id) REFERENCES longtext (id);
ALTER TABLE item
   ADD CONSTRAINT item_ord_id_fk
   FOREIGN KEY (ord_id) REFERENCES ord (id);
ALTER TABLE item
   ADD CONSTRAINT item_product_id_fk
   FOREIGN KEY (product_id) REFERENCES product (id);
ALTER TABLE warehouse
   ADD CONSTRAINT warehouse_manager_id_fk
   FOREIGN KEY (manager_id) REFERENCES emp (id);
ALTER TABLE warehouse
   ADD CONSTRAINT warehouse_region_id_fk
   FOREIGN KEY (region_id) REFERENCES region (id);
ALTER TABLE inventory
   ADD CONSTRAINT inventory_product_id_fk
   FOREIGN KEY (product_id) REFERENCES product (id);
ALTER TABLE inventory
   ADD CONSTRAINT inventory_warehouse_id_fk
   FOREIGN KEY (warehouse_id) REFERENCES warehouse (id);

set echo on
```

A. Literatura

1. Oryginalna dokumentacja firmy Oracle: *SQL Reference*
2. Oryginalna dokumentacja firmy Oracle: *PL/SQL User's Guide and Reference*
3. Bill Pribyl, Steven Feuerstein: Oracle *PL/SQL. Wprowadzenie*. Helion 2002