Paweł Rajba
pawel@ii.uni.wroc.pl
http://www.itcourses.eu/

PL/SQL

Agenda

- X

Wprowadzenie

- PL/SQL
  - rozszerzenie do SQL umożliwiające tworzenie bloków kodu z podstawowymi wyrażeniami jak pętle czy wyrażenia warunkowe
  - dalej jest możliwość wykorzystywania wszystkich dobrodziejstw SQL, co razem daje bardzo „powerfull” połączenie

Wprowadzenie

- Narzędzia
  - SQL Developer
  - SQL Developer Data Modeler
  - SQL*Plus
  - Toad
  - SQL Navigator

Komentarze

- -- komentarz
- /* inny komentarz */

Anonymous blocks

- Podstawowa jednostka w PL/SQL
- Struktura bloku
  - Deklaracja (opcjonalnie)
  - Lista instrukcji (wymagane)
  - Blok obsługi wyjątków (opcjonalnie)

Anonymous blocks

- DECLARE
  x NUMBER;
  BEGIN
  x := 10;
  EXCEPTION
  WHEN OTHERS
  THEN
  null;
  END;
  /

- BEGIN
  null;
  END;
  /

Anonymous blocks

```
DECLARE
  c NUMBER := 1;
BEGIN
  dbms_output.put_line('c = '||c);
  -- z 'c = '||C tez by bylo ok
  -- Uwaga: PL/SQL nie jest case-sensitive
EXCEPTION
  WHEN OTHERS THEN
    null;
END;
```

Anonymous blocks

- Zagnieżdżanie, wyjątki, etykiety
  - Przykład: anonymousblock.sql

Typy danych

- Rodzaje
  - Numeryczne
  - Znakowe
  - Daty i czasu
  - Boolean
- Przypisanie, stała i wartości domyślne
  - x1 NUMBER; -- domyślnie będzie NULL
  - x2 NUMBER := 10;
  - x3 NUMBER DEFAULT 7.24;
  - 4 CONSTANT NUMBER DEFAULT 8;

Typy danych

- Podtypy
  - Mogę być wbudowane lub własne
    - Wbudowane obejrzymy dalej
  - Deklaracja własnego podtypu
    - SUBTYPE <nazwa> IS <bazowy> [(constraint)] [NOT NULL]
  - Przykład
    - DECLARE
    - SUBTYPE OCENA IS PLS_INTEGER RANGE 1..6 NOT NULL;
    - x OCENA := 6;
    - BEGIN
    - dbms_output.put_line(x);
    - END;

http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/datatypes.htm#i42580

Typy danych

- Zmienna %TYPE i %ROWTYPE
  - Pozwala na pobranie typu
- Przykład

```
DECLARE
  x NUMBER(3,-2) := 12345.67;
  y x%TYPE := 12345.67;
BEGIN
  dbms_output.put_line(x||' '||y);
END;
```

Typy danych

- NUMBER
  - Zakres wartości: 1E-130 – 1E126
  - Deklaracja
    - Number(precision,scale)
      - Precision: liczba cyfr
      - Scale: miejsce zaokrąglenia; od -84 do 127
  - Bez definiowania precision i scale, NUMBER staje się zmiennoprzecinkowy

Typy danych

- Przykłady
  - NUMBER(4,2)
    - 12.34 → 12.34
    - 12.345 → 12.35
    - 123.45 → błąd (123 + 2 cyfry precision to jest 5 cyfr)
  - NUMBER(3,-2)
    - 12345,67 → 12300

Typy danych

- NUMBER, podtypy
  - DEC, DECIMAL, NUMERIC
  - REAL, DOUBLE PRECISION, FLOAT
  - INT, INTEGER, SMALLINT

Typy danych

- PLS_INTEGER / BINARY_INTEGER
  - Szybszy w obliczeniach
  - Zakres do -2147483648 do 2147483647
  - Zalecany

Typy danych

- PLS_INTEGER, podtypy
  - NATURAL → dodatnie z zerem
  - NATURALN → dodatnie z zerem, NOT NULL
  - POSITIVE → dodatnie
  - POSITIVEN → dodatnie , NOT NULL
  - SIGNTYPE → -1, 0, 1
  - SIMPLE_INTEGER → to samo, tylko NOT NULL i brak wyjątku przy overflow
    - 2147483647+1 → -2147483648

Typy danych

- BINARY_DOUBLE
  - 64bit, ~1.8E+308 – ~2.2E-308
- BINARY_FLOAT
  - 32bit, ~3.4E+38 – ~1.1E-38
- Szybsze przy obliczeniach niż NUMBER
- Overflow/under → nie ma wyjątków
- Podtypy
  - SIMPLE_FLOAT, SIMPLE_DOUBLE

Typy danych

- BINARY_DOUBLE/ BINARY_FLOAT
  - Problem z zaokrągleniami

```
DECLARE
  x BINARY_DOUBLE := 1.1;
BEGIN
  dbms_output.put_line(2*x);
END;
```

Typy danych

- BINARY_DOUBLE, BINARY_FLOAT
  - Stałe:
    - BINARY_{DOUBLE|FLOAT}_NAN
    - BINARY_{DOUBLE|FLOAT}_INFINITY
    - BINARY_{DOUBLE|FLOAT}_MAX_NORMAL
    - BINARY_{DOUBLE|FLOAT}_MIN_NORMAL
  - Przykład

```
DECLARE
 x BINARY_DOUBLE := 2;
BEGIN
 x := x/0;
 IF x = BINARY_DOUBLE_INFINITY THEN
  dbms_output.put_line(x);
 END IF;
END;
```

Typy danych

- Znakowe
  - CHAR, VARCHAR2(size [bytes | char])
  - NCHAR, NVARCHAR(size [char]) (Unicode)
- Domyślnie size=1
- Przy CHAR padding dodawany z prawej
- Maksymalny rozmiar
  - Zmienna: 32767 bajtów
  - Kolumna w tabeli: 4000 bajtów

Typy danych

- Typy daty i czasu
  - DATE – Day, Mon, Year, Hour, Minutes, Seconds
    - Domyślny format:
      SELECT * FROM nls_instance_parameters WHERE parameter='NLS_DATE_FORMAT'
    - TO_DATE(napis, format)
    - CURRENT_DATE
  - TIMESTAMP – dodatkowo tysięczne sekundy
  - TIMESTAMP WITH TIME ZONE
    - CURRENT_TIMESTAMP, TO_TIMESTAMP()

Typy danych

- BOOLEAN
  - Wartości TRUE, FALSE, NULL
  - UWAGA: wartości 0 i 1 NIE są synonimami

Pętle

- Pętla prosta
- FOR
- WHILE

- Pętle można zagnieżdżać

Pętle

- Zakończenie pętli
  - Implicit
    - Warunek dla pętli FOR i WHILE
  - Explicit
    - EXIT
    - EXIT WHEN <warunek>
    - RETURN
    - GOTO
  - Wyjątek

Pętle

- Simple loop

```
LOOP
   ...
END LOOP;
```

Pętle

- Przykład simple loop

```
DECLARE
 i NUMBER := 1;
BEGIN
 LOOP
  dbms_output.put_line(i);
  i := i+1;
  EXIT WHEN i>2;
 END LOOP;
END;
```

Pętle

- FOR

FOR i IN [REVERSE] do..do LOOP

    …

END LOOP

Zmienna i jest typu PLS_INTEGER

Pętle

- Przykład FOR

```
DECLARE
  i NUMBER;
  od NUMBER := 1;
  do NUMBER := 5;
BEGIN
  FOR I IN REVERSE od..do/2 LOOP
    dbms_output.put_line(i);
  END LOOP;
END;
```

Pętle

- CONTINUE, CONTINUE WHEN <warunek>, zagnieżdżanie

```
DECLARE
 I NUMBER;
 J NUMBER;
BEGIN
 <<outer>>
 FOR I IN 1..3 LOOP
  <<INNER>>
  FOR J IN 1..3 LOOP
   IF I+J=3 THEN
    CONTINUE OUTER;
   END IF;
   DBMS_OUTPUT.PUT_LINE(I||' '||J);
  END LOOP;
 END LOOP;
END;
```

Pętle

- WHILE

```
WHILE <warunek> LOOP
  ...
END LOOP;
```

Wyrażenia warunkowe

```
IF <warunek> THEN
...
END IF;
```

```
IF <warunek> THEN
...
ELSE
...
END IF;
```

```
IF <warunek> THEN
...
ELSEIF <warunek> THEN
...
ELSE
...
END IF;
```

Wyrażenia warunkowe

```
DECLARE
  typ_klienta VARCHAR2(20) :=
'GOLD';
  rabat NUMBER;
BEGIN
  CASE TYP_KLIENTA
    WHEN 'SILVER' THEN
      RABAT := 10;
    WHEN 'GOLD' THEN
      RABAT := 15;
    WHEN 'PLATINUM' THEN
      RABAT := 25;
    ELSE
      RABAT := 0;
  END CASE;
  dbms_output.put_line(rabat);
END;
```

```
DECLARE
  staz INTEGER := 7;
  rabat NUMBER;
BEGIN
  CASE
    WHEN staz >= 1 AND staz <=2
THEN
      RABAT := 10;
    WHEN staz < 5 THEN
      RABAT := 15;
    WHEN staz >= 5 THEN
      RABAT := 25;
    ELSE
      RABAT := 0;
  END CASE;
  dbms_output.put_line(rabat);
END;
```

Wyrażenia warunkowe

```
DECLARE
  typ_klienta VARCHAR2(20) :=
'GOLD';
  rabat NUMBER;
BEGIN
  RABAT :=
    CASE TYP_KLIENTA
      WHEN 'SILVER' THEN 10
      WHEN 'GOLD' THEN 15
      WHEN 'PLATINUM' THEN 25
      ELSE 0
    END;
  dbms_output.put_line(rabat);
END;
```

```
DECLARE
  staz INTEGER := 7;
  rabat NUMBER;
BEGIN
  rabat :=
    CASE
      WHEN STAZ >= 1 AND STAZ
<=2 THEN 10
      WHEN STAZ < 5 THEN 15
      WHEN STAZ >= 5 THEN 25
      ELSE 0
    END;
  dbms_output.put_line(rabat);
END;
```

Procedury

- Uprawnienia
  - CREATE PROCEDURE, CREAT ANY PROCEDURE
  - ALTER ANY PROCEDURE, EXECUTE
- Tworzenie

```
CREATE [OR REPLACE] PROCEDURE nazwa IS|AS
   <deklaracje>
BEGIN
   ...
EXCEPTION
   ...
END;
```

Procedury

```
CREATE TABLE LICZNIK (WARTOSC NUMBER(8));
insert into licznik values(1);

CREATE OR REPLACE PROCEDURE podnies_licznik AS
BEGIN
  UPDATE LICZNIK SET WARTOSC = WARTOSC+1;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
  ROLLBACK;
END;

EXEC PODNIES_LICZNIK;

SELECT * FROM LICZNIK;
```

Rodzaje kompilacji

- PL_SQL_CODE_TYPE
  - INTERPRETED
  - NATIVE
- Dostępne od wersji 11g
  - ALTER SESSION SET PL_SQL_CODE_TYPE=NATIVE
  - ALTER PROCEDURE <nazwa> COMPILE PL_SQL_CODE_TYPE=NATIVE

Poziom optymalizacji

- PLSQL_OPTIMIZE_LEVEL
  - 0 Pre 10g Optimization
  - 1 Removed Unnecessary Computations
  - 2 Code Refactoring
  - 3 Code Inlining
- Jak zrobić?
  - ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=2
  - SELECT PLSQL_OPTIMIZE_LEVEL, PLSQL_CODE_TYPE FROM ALL_PLSQL_OBJECT_SETTINGS WHERE NAME=<nazwa>

Kompilacja dla debugowania

- INTERPRETED
- Tylko na środowiskach nieprodukcyjnych
- PLSQL_DEBUG
- ALTER PROCEDURE <nazwa> COMPILE DEBUG;
- ALTER SESSION SET PLSQL_DEBUG=FALSE;

Błędy i ostrzeżenia

- Przy kompilacji mamy błędy
  - Kiedy pomylimy nazwę obiektu (np. tabeli)
  - składniowe
- Ostrzeżenia
  - Performance, Informational, Severe
    - np. Nieosiągalny kod, nieużywana zmienna
  - Ostrzeżenia można traktować następująco:
    - enable,
    - disable,
    - error

Procedury

- Wywoływanie
  - CALL nazwa()
  - EXEC nazwa
  - EXECUTE nazwa
  - BEGIN
    nazwa;
    END;

Funkcje

- Wbudowane i użytkownika
- Sporo funkcji wbudowanych
  - ROUND, CEIL, LPAD, LTRIM
  - DBMS_RANDOM.VALUE(1,100)
  - NVL(napis, zamiennik)
    - Jeśli napis jest NULL, zostanie zwrócony zamiennik

Funkcje

- Tworzenie funkcji

```
CREATE [OR REPLACE] FUNCTION nazwa
   RETURN <typ_danych> IS|AS
   <deklaracje>
BEGIN
   ...
RETURN <wynik>;
EXCEPTION
   ...
END;
```

Funkcje

- Wywoływanie
  - Zmienna := nazwa_funkcji;
  - SELECT nazwa_funkcji FROM DUAL;

Funkcje deterministyczne

- Test

```
CREATE [OR REPLACE] FUNCTION nazwa
   RETURN <typ_danych> DETERMINISTIC IS|AS
   <deklaracje>
BEGIN
   ...
RETURN <wynik>;
EXCEPTION
   ...
END;

SELECT EMPLOYEE_ID, CZY_DUZO_ZARABIA(SALARY) FROM
EMPLOYEES
```

Funkcje deterministyczne

```
CREATE OR REPLACE FUNCTION czy_duzo_zarabia(pensja
NUMBER)
   RETURN VARCHAR DETERMINISTIC AS
BEGIN
  dbms_output.put_line('X');
  IF PENSJA <5000 THEN
    RETURN 'duzo';
  ELSE
    RETURN 'malo';
  END IF;
END;

SELECT EMPLOYEE_ID, CZY_DUZO_ZARABIA(SALARY)
FROM EMPLOYEES;
```

Parametry do procedur i funkcji

- Tryby przekazywania parametrów
  - IN
  - OUT
  - IN OUT
- CREATE … nazwa(p1, p2,…,pN)
  - gdzie pi to <nazwa> <tryb> <typ> [{:=|DEFAULT} wi]
- Domyślnie
  - IN – przez referencję
  - OUT, IN OUT – przez wartość

Parametry do procedur i funkcji

- NOCOPY hint
  - Jest to „sugestia” dla kompilatora, żeby przekazywać zmienną przez referencję
  - Ma sens dla OUT i IN OUT
  - Jest cała lista sytuacji, w których kompilator zignoruje sugestię

http://docs.oracle.com/cd/E11882_01/appdev.112/e25519/parameter_declaration.htm#LNPLS1271

Parametry do procedur i funkcji

- Przekazywanie parametrów
  - Pozycyjne
    - Tradycyjne
  - Poprzez nazwę
    - Podajemy nazwę parametru przy wywołaniu i wartość

    procedura(imie => 'Jan', nazwisko=>'Kowalski')
  - Kombinacja
    - Na początku tradycyjne i nazwę

    procedura('Jan', nazwisko=>'Kowalski')