

KBD2 — Komercyjne bazy danych

dr inż. Tomasz Traczyk

3. Język PL/SQL

Listopad 2007

Copyright © Tomasz Traczyk
Instytut Automatyki i Informatyki Stosowanej Politechniki Warszawskiej
Materiały dydaktyczne przeznaczone są wyłącznie do indywidualnego użytku studiujących.
Rozpowszechnianie kopii bez pisemnej zgody autora jest zabronione.

Streszczenie

Wykład przedstawia język PL/SQL stosowany w Oracle do programowania proceduralnego po stronie serwera i po stronie klienta — w aplikacjach tworzonych za pomocą narzędzi Forms/Reports.

Język PL/SQL

- Proceduralny język programowania
- Niezle zintegrowany z SQL
- Zastosowanie
 - Programowanie proceduralne po stronie DBMS: wyzwalacze, składowane w bazie danych procedury, funkcje i pakiety
 - Pomocnicze programowanie w narzędziach Forms i Reports
- Cechy
 - całkowita przenośność
 - brak transmisji sieciowych
 - interpretowany - spore zużycie zasobów serwera
 - istnieje możliwość kompilacji do kodu maszynowego

Podstawowe konstrukcje

- Język strukturalny
- Budowa blokowa
- Format
- Instrukcja podstawienia
 - postać: `zmienna:= wyrażenie;`
 - niezbędne konwersje typów — automatyczne
- Bloki
 - bloki nazwane, czyli moduły
 - * wyzwalacze
 - * procedury
 - * funkcje
 - * pakiety
 - bloki anonimowe
 - bloki podrzędne

Budowa bloku

```
<<etykieta_bloku>>  
DECLARE  
    deklaracje;  
BEGIN  
    ciało;  
EXCEPTION  
    obsługa wyjątków;  
END;
```

- Etykieta, części DECLARE i EXCEPTION — opcjonalne
- W głównym bloku słowo DECLARE pomija się
- Zagnieżdżanie bloków

Dane

- Stałe i zmienne muszą być deklarowane
- Deklaracje stałych: nazwa_stalej CONSTANT typ := wyrażenie;
- Deklaracje zmiennych: nazwa_zmiennej typ [NOT NULL [:= wyrażenie]];
 - zamiast := można użyć DEFAULT
 - zmienna niezainicjowana ma wartość NULL
 - deklarowanie zmiennej o typie zgodnym z kolumną: nazwa_zmiennej nazwa_tabeli.nazwa_kolumny%TYPE
 - deklarowanie zmiennej o typie identycznym wcześniej zadeklarowaną zmienną: nazwa_nowej_zmiennej nazwa_starej_zmiennej%TYPE

Proste typy danych

- Wszystkie typy dopuszczalne dla kolumn
- BINARY_INTEGER – „szybki” typ numeryczny 32 bitowy
 - NATURAL – liczby nieujemne
 - POSITIVE – liczby dodatnie
- BOOLEAN – typ logiczny (stałe TRUE i FALSE)

Złożone typy danych

Tablice

- Tablice *index-by*
 - deklaracja typu: `TYPE nowy_typ IS TABLE OF typ_prosty INDEX BY BINARY_INTEGER;`
 - deklaracja zmiennej — przy użyciu nazwy zdefiniowanego typu
 - odwołanie do komórki `zmienna_tablicowa(indeks)`
- Inne typy tablicowe: *nested tables* i *varrays* → później

Rekordy

- Deklaracja: `TYPE nowy_typ IS RECORD (lista_deklaracji_pól);`
- Deklaracje pól — jak deklaracje zmiennych
- Dozwolone zagnieżdżanie pól wcześniej zdefiniowanych typów rekordowych
- Deklaracja zmiennej — przy użyciu nazwy zdefiniowanego typu
- Deklaracja rekordu o strukturze wiersza tabeli: `nazwa_zmiennej nazwa_tabeli%ROWTYPE;`
- Odwołania do pól: `nazwa_zmiennej_rekordowej.nazwa_pola`

```
DECLARE
TYPE rekord IS RECORD (
    ocena NUMBER(3,1),
    wyklawowca wyklawowcy%ROWTYPE
);
zapis rekord;
BEGIN
zapis.wyklawowca.id_wyklawowcy := 1;
...
END;
```

deklaracja typu
zagnieżdżona struktura rekordowa
deklaracja zmiennej typu rekordowego
odwołanie do pola zagnieżdżonego rekordu

Instrukcje sterujące

- Instrukcja pusta: `NULL;`
- Instrukcja warunkowa:
`IF warunek THEN`
instrukcje
`ELSIF`
instrukcje
`ELSE`
instrukcje
`END IF;`
- Pętle
 - z każdej pętli możliwy wyskok instrukcją `EXIT [etykieta] [WHEN warunek];`
 - etykiety umożliwia wyskok przez kilka poziomów, do pętli poprzedzanej: `<<etykieta>>`
- Pętla bezwarunkowa:
`LOOP`
instrukcje
`END LOOP;`
- Pętla warunkowa:
`WHILE warunek LOOP`
instrukcje
`END LOOP;`
- Pętla wyliczana:
`FOR licznik IN początek..koniec LOOP`
instrukcje
`END LOOP;`

Obsługa wyjątków

- Propagacja wyjątków
 - nieobsłużone w bloku — przekazywane do bloku zewnętrznego
 - nieobsłużone w żadnym bloku — przekazywane do środowiska wykonania PL/SQL
 - po obsłużeniu wyjątku wykonanie bloku kończy się
 - wyjątek już obsłużony można przekazać dalej: RAISE;
- Sekcja EXCEPTIONS:
EXCEPTION
 - WHEN wyjątek 1 THEN
 obsługa wyjątku 1
 - WHEN wyjątek 2 THEN
 obsługa wyjątku 2
 - ...
 - WHEN OTHERS THEN
 obsługa pozostałych wyjątkówEND;

- Nazwy wyjątków
 - predefiniowane
 - deklarowane: wyjątek EXCEPTION;
- Wywoływanie wyjątków
 - predefiniowany — wywoływany automatycznie gdy znajdzie przypisany mu błąd
 - deklarowany
 - * wywoływany programowo: RAISE wyjątek;
 - * wywoływany automatycznie po przypisaniu do nienazwanych błędów:
PRAGMA EXCEPTION_INIT(nazwa, numer_błędu);
(wyjątek musi być wcześniej zadeklarowany)
 - * wywoływanie błędu o określonym numerze:
raise_application_error(numer_błędu, komunikat);
 - numer musi być z zakresu -20999..-20000
 - najczęściej używane w wyzwalaczach

Wyjątki predefiniowane

LOGIN_DENIED	Nieudane przyłączenie do bazy	ORA-01017
NOT_LOGGED_ON	Nie przyłączono się do bazy	ORA-01012
NO_DATA_FOUND	Instrukcja SELECT INTO nie zwróciła żadnego wiersza	ORA-01403
TOO_MANY_ROWS	Instrukcja SELECT INTO zwróciła więcej niż jeden wiersz	ORA-01422
DUP_VAL_ON_INDEX	Próba naruszenia ograniczenia unikalnej wartości (klucz główny, unikalny lub unikalny indeks)	ORA-00001
INVALID_CURSOR	Nielegalna operacja na kursorze	ORA-01001
VALUE_ERROR	Błąd wartości, np. niemożliwa konwersja	ORA-06502
ZERO_DIVIDE	Dzielenie przez zero	ORA-01476

Podprogramy

FUNCTION nazwa (parametry) RETURN typ IS deklaracje;	PROCEDURE nazwa (parametry) IS deklaracje;
BEGIN	BEGIN
...	...
RETURN wartość;	END;
END;	

- Parametry:
 - nazwa [IN|OUT|IN OUT] typ [:=wartość] (domyślny jest tryb IN)
- Podprogramy lokalne – definicja *na końcu*
- sekcji deklaracji
 - *Forward declaration*
 - Przeciążanie nazw podprogramów (lokalnych i w pakietach):

Pakiety

- **Specyfikacja:**
PACKAGE nazwa IS
 deklaracje obiektów publicznych;
 deklaracje podprogramów publicznych;
END [nazwa];
- **Ciało:**
PACKAGE BODY nazwa IS
 deklaracje obiektów prywatnych;
 podprogramy;
BEGIN
 inicjalizacja;
END [nazwa];
- **Zalety pakietów**
- **Inne cechy**

Użycie podprogramów i pakietów

- **Wywołanie podprogramów**
 - w PL/SQL: nazwa(parametry); (jeśli nie ma parametrów to nie pisze się także nawiasów)
 - w SQL*Plus: EXECUTE nazwa(parametry);
 - dla obiektów z pakietu: pakiet.nazwa_obiektu[(parametry)]
 - można używać notacji z nazwami parametrów: nazwa_parametru=>wartość
 - można pominąć parametry mające wartości domyślne (dla notacji pozycyjnej tylko końcowe)
- **Cechy użycia podprogramów**
 - rekurencja dozwolona
 - parametry aktualne dla typów OUT i IN OUT muszą być zmiennymi
 - w zapytaniach SQL można używać funkcji PL/SQL (o ile nie zmieniają danych!)
 - funkcje z pakietów używane w SQL muszą być opatrzone dyrektywą (po deklaracji)
PRAGMA restrict_references(nazwa,WNDS,WNPS)

PL/SQL w bazie danych

Przechowywanie w bazie danych

- Podprogramy przechowywane w postaci źródłowej i skompilowanej
- Istnieje możliwość „zakodowania” wersji źródłowej

Prawa dostępu

- Prawa systemowe do tworzenia i modyfikacji: CREATE PROCEDURE
- Prawa obiektowe do wywoływania: EXECUTE
- Działanie procedur/funkcji: z prawami twórcy (tylko przyznanymi *bezpośrednio*)

Kompilacja

Kompilacja

- Do *p*-kodu
 - samoczynnie przy ładowaniu podprogramu
 - ręcznie:

```
ALTER PROCEDURE|FUNCTION|TRIGGER nazwa COMPILE;  
ALTER PACKAGE nazwa COMPILE PACKAGE|BODY;
```
- Do kodu maszynowego (*native execution*)

```
ALTER SESSION SET PLSQL_CODE_TYPE='NATIVE';
```

Zależności i rekompilacja

- Zależność podprogramów (perspektywa USER_DEPENDENCIES)
- Efekty zmian w łańcuchu zależności

Tworzenie kodu PL/SQL na serwerze

```
CREATE [OR REPLACE] PROCEDURE|FUNCTION nazwa... IS|AS...
CREATE [OR REPLACE] PACKAGE [BODY] nazwa ... IS|AS...
DROP PROCEDURE|FUNCTION|PACKAGE [BODY] nazwa;
```

W SQL*Plus definicja musi być zakończona linią zawierającą jedynie ukośnik w 1 kolumnie

Uruchamianie programów PL/SQL

- Znajdowanie błędów kompilacji w SQL*Plus:
 - L – listing kodu (z numerami wierszy)
 - SHOW ERRORS – opis błędów
- Śledzenie wykonania kodu w SQL*Plus:
 - SET SERVEROUTPUT ON
 - w procedurach używać pakietu DBMS_OUTPUT:
DBMS_OUTPUT.PUT_LINE(komunikat);
- Użycie zmiennych w SQL*Plus
 - deklarowanie: VAR zmienna typ
 - sprawdzanie: PRINT zmienna
- Program SQL Developer
 - w miarę porządnym debugger do PL/SQL

Obsługa danych w PL/SQL

Zanurzony (embedded) SQL

- Zdania SQL zanurzone w kodzie w języku proceduralnym
- Możliwe przekazywanie danych między zapytaniami SQL i zmiennymi języka proceduralnego
- Problemy związane z odmiernością języków (*impedance mismatch*)
- Specjalne konstrukcje (kursory) do przetwarzania danych wiersz-po-wierszu

SQL w PL/SQL

- Dobra zgodność typów
 - typy z b.d. dopuszczone w PL/SQL
 - możliwe definiowanie zmiennych na podst. typów kolumn
- Zapytania SQL umieszczane wprost w kodzie PL/SQL (bez specjalnych wyróżnień)
- Obsługa błędów b.d. przez mechanizm wyjątków
- Specjalne typy danych (kursory) i pętli do obsługi SQL

Zapytania ze zmiennymi związanymi

- Zmienne PL/SQL wykorzystywane w zapytaniach SQL zanurzonych w kodzie (nazwy zmiennych *bez* specjalnych oznaczeń)
- Miejsce zmiennych: w warunkach oraz jako źródło i przeznaczenie danych
- Pobieranie danych do zmiennych: `SELECT ... INTO lista_zmiennych FROM...`;
 - listy kolumn i zmiennych dopasowane co do liczby i typu
 - takie zapytanie musi zwracać *dokładnie jeden* wiersz
 - odmienny wynik powoduje wyjątki: `NO_DATA_FOUND` lub `TOO_MANY_ROWS`

Modyfikacja danych

- Modyfikacja zdaniami `INSERT`, `UPDATE`, `DELETE`
- Zmienne PL/SQL używane w warunkach i jako źródło danych w modyfikacjach
- Można używać `SELECT ..FOR UPDATE...[NOWAIT]`
- Specjalna konstrukcja `WHEN CURRENT OF` do modyfikacji danych za pomocą kursorów
- Kontrola transakcji:
 - `COMMIT`, `ROLLBACK`,
 - `SAVEPOINT nazwa`, `ROLLBACK TO nazwa`
 - `SET TRANSACTION READ ONLY`;
 - domyślne punkty kontrolne instrukcji DML

Kursory

- Służą do dostępu do wyniku zapytania wiersz po wierszu
- Operacje
 - otwarcie (operacje na nieotwartym kursorze dają wyjątek `INVALID_CURSOR`)
 - `FETCH` – pobranie wiersza
 - zamknięcie
- Atrybuty kursora
 - `cursor%ISOPEN`
 - `cursor%FOUND`, `cursor%NOTFOUND` (`NULL` przed pierwszym pobraniem)
 - `cursor%ROWCOUNT` – liczba wierszy dotychczas *pobranych*
- Rodzaje cursorów
 - jawne – obiekt deklarowany i/lub obsługiwany jawnie
 - niejawne – tworzone przez system do realizacji SQL (dostęp do atrybutów: `SQL%atrybut`)

Kursory jawne

- Deklaracja: `CURSOR nazwa_kursora[(parametry_formalne)] IS SELECT...`;
 - parametry mogą być wykorzystywane w warunkach
 - postać definicji parametru: `nazwa_parametru typ [DEFAULT wartość]`
- Wykorzystanie:
 - otwarcie: `OPEN nazwa_kursora[(parametry_aktualne)];`
 - pobieranie wierszy: `FETCH nazwa_kursora INTO lista_zmiennych;`
 - stwierdzenie końca danych, np.:
`EXIT WHEN nazwa_kursora%NOTFOUND OR cursor%NOTFOUND IS NULL;`
 - zamknięcie: `CLOSE nazwa_kursora;`
- W obsłudze kursora można stosować konstrukcje:
`UPDATE ... WHERE CURRENT OF nazwa_kursora;`
`DELETE ... WHERE CURRENT OF nazwa_kursora;`
 - zapytanie definiujące kursor musi być „modyfikowalne”
 - kursor musi być zadeklarowany jako `...FOR UPDATE`

Kursory w pakietach

- Kursor publiczny w specyfikacji pakietu wymaga specjalnej deklaracji:
`CURSOR nazwa [parametry_formalne] RETURN typ`
 - typ rekordowy, np. `%ROWTYPE`
- W ciele pakietu — zwykła definicja

Zmienne kursorowe

- Są to wskaźniki do obszaru danych
- Definiowanie:
 - deklaracja typu: `TYPE nazwa IS REF CURSOR [RETURN typ]`
 - deklaracja zmiennej z użyciem tego typu
 - nie można tak definiować publicznych zmiennych pakietu
 - zmienne kursorowe nie mogą mieć parametrów
- Użycie:
 - otwarcie: `OPEN nazwa FOR SELECT...;`
 - pobieranie danych: `FETCH nazwa INTO...;`
 - zamknięcie: `CLOSE nazwa`
 - zmienne kursorowe mogą być przekazywane jako parametry do/z podprogramów
 - poszczególne operacje mogą być wykonywane w różnych podprogramach

Pętla FOR z kursorem

- Pętla z kursorem niejawnym
FOR zmienna IN (SELECT ...) LOOP
...
END LOOP;
- Pętla z kursorem jawnym
FOR zmienna IN nazwa_kursora [(parametry_aktualne)] LOOP
...
END LOOP;
- zmienna jest deklarowana *domyślnie* i ma typ rekordowy odpowiedni do zapytania
- Konstrukcja realizuje niejawnie OPEN, FETCH i CLOSE
- Dostęp do danych wewnątrz pętli: zmienna.nazwa_kolumny
- Można używać aliasów dla kolumn

Wyzwalacze

- Aktywne reguły w bazie danych
- Wykonywane automatycznie przez serwer — nie ma możliwości omińnięcia
- Rodzaje wyzwalaczy
 - Okoliczność zadziałania
 - * BEFORE/AFTER
 - * DELETE/UPDATE/INSERT (można łączyć przez OR)
 - Zasięg działania
 - * dla całego zdania DML (*statement level*)
 - * FOR EACH ROW
- Zastosowanie

Ograniczenia wyzwalaczy

- Wyzwalacze FOR EACH ROW oraz wszystkie wywołane *pośrednio* przez DELETE CASCADE) nie mogą:
 - czytać ani pisać w *mutating tables*^a — tabelach których zmiana wyzwoliła działanie
 - modyfikować kluczy w *constraining tables*^b — tabelach, które muszą być wykorzystane do sprawdzenia integralności referencyjnej po zmianie, która wyzwoliła działanie
- Wyzwalacze na poziomie instrukcji nie mają dostępu do kontekstu
- Wyzwalacze nie mogą (także pośrednio) wykonywać COMMIT, ROLLBACK ani SAVEPOINT

^aZ wyjątkami opisanymi dalej

^bDotyczy starszych wersji bazy Oracle

Tworzenie wyzwalacza

```
CREATE [OR REPLACE] TRIGGER nazwa
BEFORE|AFTER operacja ON tabela
[FOR EACH ROW
WHEN (warunek)]
DECLARE
...
BEGIN
...
END;
/
```

- Szczegóły:
 - można uściślić: UPDATE OF kolumny ON tabela
 - rodzaj zmiany może być stwierdzony za pomocą predykatów:
IF INSERTING|UPDATING['kolumny']|DELETING THEN
- Czasowe wyłączenie: ALTER TRIGGER nazwa ENABLE|DISABLE;
- Prawa dostępu
 - tworzenie i modyfikacja: CREATE TRIGGER i ALTER do tabeli
 - działanie: z prawami twórcy (przyznanymi *bezpośrednio*)

Wyzwalacze FOR EACH ROW

- Dostęp do kontekstu: do zawartości bieżącego wiersza
- Brak dostępu do innych wierszy zmieniającej się tabeli, chyba że jest to wyzwalacz BEFORE|AFTER INSERT FOR EACH ROW wywołony przez wstawianie *pojedynczego* wiersza
- Odwołanie do wartości przed i po zmianach:
:OLD.nazwa_kolumny i :NEW.nazwa_kolumny
- Uwaga na NULL dla nieokreślonych wartości (np. :OLD przy INSERT)
- W wyzwalaczach BEFORE FOR EACH ROW można modyfikować bieżący wiersz przez podstawienie pod :NEW
- Warunek WHEN dodatkowo ogranicza okoliczności zadziałania
 - tu *nie* używa się dwukropków przed OLD i NEW
 - wartość NULL jest traktowana tak jak fałsz

Przykład 1

```
CREATE OR REPLACE TRIGGER wyk_ins_trg
BEFORE INSERT ON wykladowcy
FOR EACH ROW
BEGIN
    SELECT wykladowcy_seq.NEXTVAL INTO :NEW.id_wykladowcy FROM dual;
END;
```

Przykład 2

```
CREATE OR REPLACE TRIGGER wyk_upd_trg
BEFORE INSERT OR UPDATE OF nazwisko ON wykladowcy
FOR EACH ROW
WHEN ( NVL(OLD.nazwisko,'-') <> NEW.nazwisko )
BEGIN
    :NEW.nazwisko := UPPER(:NEW.nazwisko);
END;
```

wyzwała zmiana tylko tej kolumny
tylko gdy kolumna rzeczywiście zmieniona
zapis przez podstawienie pod :NEW

Przykład 3

```
CREATE OR REPLACE TRIGGER prz_upd_trg
BEFORE INSERT OR UPDATE OR DELETE ON przedmioty
DECLARE
    errno CONSTANT number := -20000;
    dummy wykladowcy.id_wykladowcy%TYPE;
BEGIN
    SELECT id_wykladowcy INTO dummy FROM wykladowcy WHERE nazwisko=USER;
EXCEPTION
WHEN NO_DATA_FOUND THEN
    RAISE_APPLICATION_ERROR(errno,'Operacja niedozwolona');
END;
```

definicja stałej

Funkcje i procedury wbudowane

- Dostępne są funkcje standardowe SQL
- SQLCODE, SQLERRM – zwracają numer i opis błędu
- Procedury i funkcje serwera pogrupowane w pakiety
- Niektóre pakiety są częściami opcji serwera
- Opisów należy szukać w \$ORACLE_HOME/rdbms##/admin, w plikach dbms*.sql lub utl*.sql

Pakiety serwera

STANDARD	procedury standardowe, przy wywołaniu <i>nie trzeba</i> podawać nazwy pakietu	raise_application_error
DBMS_SESSION	obsługa sesji	set_role
DBMS_DDL	kompilacja, tworzenie statystyk	alter_compile, analyze_object
DBMS_UTILITY	kompilacja i statystyki, obsługa stosu błędów	
DBMS_TRANSACTION	obsługa transakcji	read_only, use_rollback_segment, begin_discrete_transaction
DBMS_APPLICATION_INFO	rejestracja informacji o aplikacji i jej stanie w słowniku systemowym	
DBMS_SPACE	informacje o wolnej przestrzeni	
DBMS_ROWID	operacje na ROWID	
DBMS_LOCK	obsługa blokad	request, release

DBMS_ALERT	obsługa alertów, operacje jak na semaforach (działają w chwili commitu), wraz z sygnałem można przesłać komunikat	signal, waitone, waitany
DBMS_PIPE	przekazywanie danych przez potoki (w jednej instancji b.d.)	pack_message, send_message, receive_message, next_item_type, unpack_message
DBMS_OUTPUT	komunikaty tekstowe do aplikacji zewnętrznych	put, new_line, put_line, get_line, get_lines
DBMS_DESCRIBE	informacje o argumentach podprogramów	
DBMS_SHARED_POOL	obsługa SGA w celu zmniejszenia fragmentacji pamięci	
DBMS_SNAPSHOT	obsługa migawek	
DBMS_JOB	automatyczne wywoływanie podprogramów na serwerze	
DBMS_SQL	dynamiczny SQL	

DBMS_RANDOM	generator liczb pseudolosowych	initialize, random
DBMS_LOB	obsługa typów LOB	loadfromfile, compare, copy, instr, substr
UTL_RAW	obsługa typów RAW	
UTL_FILE	operacje plikowe na serwerze	
UTL_HTTP	dostęp do usług zewnętrznych przez protokół http	request