Wydział Elektroniki
i Technik Informacyjnych
POLITECHNIKA WARSZAWSKA

Bazy Danych 1

edycja 21L

Laboratorium 8

Wprowadzenie i przebieg laboratorium

✏️ Język PL/SQL

- Bloki
- Zmienne, typy danych, przypisania
- Przepływ sterowania
- Funkcje
- Procedury

📌 PL/SQL - Language Reference

Język PLSQL

- PL/SQL to język programowania zapewniający proceduralne rozszerzenie SQLa
- PL/SQL dostępny jedynie w bazach Oracle
- Język wysoko-wydajny, silnie zintegrowany z bazą danych

- PL/SQL ma strukturę blokową
- komentarze (-- oraz /* */)
- definicje bloków PL/SQL kończy się ukośnikiem /

Is PL SQL good for Career?

PL SQL is an integrated and high-performance database language that can work well with other languages like C++, Java, and C#. However, if you want to write a code that is going to interact with **Oracle** database, there is no better language than **PL SQL** for this job.

Is PL SQL Dead?

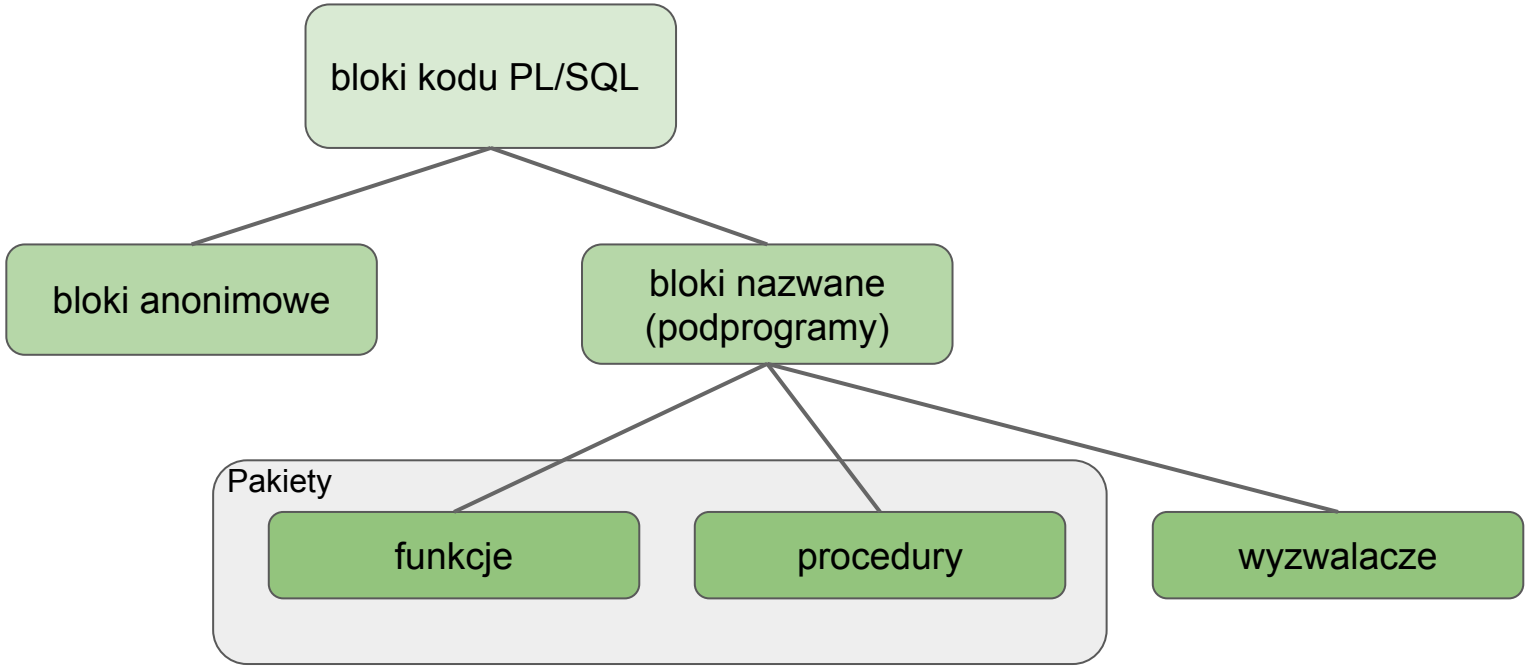
The answer is that **PL/SQL** is not growing, but not going away either. Because it is used in the **Oracle** database, and the **Oracle** database is a fixture of enterprise systems world-wide, it will outlive you. High-performance batch processing has to happen close to the data, so **PL/SQL** will continue to rule in this area. 1 sie 2016

How long will it take to learn PL SQL?

about two to three weeks

It **should take** an average learner about two to three weeks to master the basic concepts of **SQL** and start working with **SQL** databases.

Język PLSQL


bloki kodu PL/SQL
bloki anonimowe
bloki nazwane
(podprogramy)
Pakiety
funkcje
procedury
wyzwalacze

Blok anonimowy

Nie posiada nazwy, nie jest przechowywany w bazie danych. (Może być zapisany jako skrypt poza bazą)
Jest kompilowany za każdym razem przed wykonaniem.

```
[DECLARE]
  -- opcjonalny blok deklaracji zmiennych
BEGIN
  -- obowiązkowy blok wykonywalny
  --- (może mieć zagnieżdżenia)
[EXCEPTION]
  -- opcjonalny blok obsługi wyjątków
END;
```

Polecenia i kontrola przepływu

W PL/SQL można wykonywać następujące polecenia:

- polecenia SQL
- polecenia nie-SQL
  - operacje na zmiennych
  - wywołanie procedur/funkcji
  - instrukcja NULL

Dostępne są klasyczne mechanizmy kontroli przepływu sterowania:

- instrukcje warunkowe (IF, CASE)
- pętle (WHILE, LOOP, FOR)
- skok (GOTO)

Blok anonimowy [1] - ćwiczenia

1. Napisz prosty blok anonimowy zawierający blok wykonawczy z instrukcją NULL. Uruchom ten program.
2. Zmodyfikuj program powyżej i wykorzystaj procedurę dbms_output.put_line przyjmującą jako parametr łańcuch znakowy do wyświetlenia na konsoli. Uruchom program i odnajdź napis.
3. Napisz blok anonimowy który doda do tabeli region nowy rekord (np. ‘Oceania’). Uruchom program i zweryfikuj działanie.
4. Napisz blok anonimowy, który wygeneruje błąd (RAISE_APPLICATION_ERROR przyjmującą 2 parametry: kod błędu oraz wiadomość)

Zmienne

Zmienne muszą zostać zadeklarowane w bloku `DECLARE` przed użyciem. Zmienne mają typ danych i można je zainicjalizować. Bez inicjalizacji ich wartość to `NULL`. Wartość zmiennych może ulec zmianie. Można użyć słowa kluczowego `CONSTANT` / `DEFAULT`.

```
identifier [CONSTANT] datatype [NOT NULL] [:= expr |DEFAULT expr];
```

Typy zmiennych:

- skalarne (`VARCHAR2, CHAR, NUMERIC, DATE, TIMESTAMP, BOOLEAN...`)
- złożone (`RECORD, VARARRAY...`)
- własne (zdefiniowane typy danych)

W deklaracji typów można odwołać się do typu kolumny z tabeli (`%TYPE`) lub do typu wierszowego (`%ROWTYPE`) (typy zakotwiczone).

Zmienne - przykład

```
DECLARE
    v_id       NUMBER := 102;
    v_name     VARCHAR2 (50);
    v_surname  employees.surname%TYPE;
    v_employee employees%ROWTYPE;
    c_magic CONSTANT NUMBER := 10;
BEGIN
    dbms_output.put_line('Employee with id '|| v_id|| ' has name '|| v_name || ' '|| v_surname);

    SELECT name, surname
    INTO   v_name, v_surname
    FROM   employees
    WHERE  employee_id = v_id;

    dbms_output.put_line('Employee with id '|| v_id|| ' has name '|| v_name || ' '|| v_surname);

    v_id := v_id + length(v_surname) + c_magic;

    SELECT *
    INTO   v_employee
    FROM   employees
    WHERE  employee_id = v_id;

    dbms_output.put_line('Employee with id '|| v_id|| ' has name '|| v_employee.name || ' '|| v_employee.surname);

    INSERT INTO countries(country_id,name,capital) VALUES (129,'Islandia','Reykjavík');
END;
/
```

Instrukcje warunkowe - składnia

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

```
CASE [ expression ]
 WHEN val_1 THEN statement_1
  WHEN val_2 THEN statement_2
  WHEN val_n THEN statement_n
  ELSE statement_else
END;
```

```
CASE
 WHEN condition_1 THEN statement_1
  WHEN condition_2 THEN statement_2
  WHEN condition_n THEN statement_n
  ELSE statement_else
END;
```

Pętle - składnia

```
LOOP
    statements;
    EXIT [WHEN condition];
END LOOP;
```

pętla LOOP

```
WHILE condition
    statements;
END LOOP;
```

pętla WHILE

```
FOR counterIN [REVERSE] lower_bound..upper_bound LOOP
    statements;
END LOOP;
```

pętla FOR

Blok anonimowy [2] - ćwiczenia

1. Napisz blok anonimowy który będzie korzystał z dwóch zmiennych (v_min_sal oraz v_emp_id) i który będzie wypisywał na ekran imię i nazwisko pracownika o wskazanym id tylko jeśli jego zarobki są wyższe niż v_min_sal.

Nazwane bloki kodu (podprogramy)

- W PL/SQL można zaimplementować nazwane bloki kodu:
  - procedury,
  - funkcje,
  - wyzwalacze.
- Nazwane bloki kodu są jednokrotnie kompilowane (przy tworzeniu) i przechowywane w bazie danych.
- Podprogramy mogą być wołane po nazwie.
- Procedury i funkcje mogą być elementami pakietów.

Funkcje

- Nazwany blok PL/SQL.
- Może przyjmować parametry.
- Może zwracać wynik poprzez parametry wyjściowe.
- Musi zwraca pojedynczą wartość (słowo kluczowe RETURN).
- Funkcja jest kompilowana i przechowywana w bazie.
- Funkcja może być wywołana w poleceniu SQL, w innej funkcji, w procedurze, w bloku anonimowym.

⚠️ Nie każda funkcja może być wywołana w SQL (funkcje zwracające typ Boolean, lub wykonujące operacje DML nie mogą być wywołane z SQL).

Funkcje

```
CREATE [ OR REPLACE] FUNCTION name [(parameter1 [mode1] datatype1, ...)]
  RETURN datatype
IS | AS
  [local_variable_declarations;...]
BEGIN
     --actions;
  RETURN expression;
[EXCEPTION]
  WHEN
  EXCEPTION
    --actions;
END [name];
```

Funkcje - przykład

```
CREATE OR replace FUNCTION calculate_seniority_bonus(p_id NUMBER)
RETURN NUMBER
AS
  v_age           NUMBER;
  v_yrs_employed  NUMBER;
  v_birth_date    DATE;
  v_date_employed DATE;
  v_salary        NUMBER;
  v_bonus         NUMBER := 0;
  c_sal_multiplier CONSTANT NUMBER := 2;
  c_age_min CONSTANT NUMBER := 30;
  c_emp_min CONSTANT NUMBER := 3;
BEGIN
    SELECT birth_date,date_employed, salary
    INTO   v_birth_date, v_date_employed, v_salary
    FROM   employees
    WHERE  employee_id = p_id;

    v_age := extract (year FROM SYSDATE) - extract (year FROM v_birth_date);
    v_yrs_employed := extract (year FROM SYSDATE) - extract (year FROM v_date_employed);

    IF v_age > c_age_min AND v_yrs_employed > c_emp_min THEN
      v_bonus := c_sal_multiplier * v_salary;
    END IF;
    RETURN v_bonus;
END;
/
```

Zaimplementuj funkcję, która wylicza dodatek stażowy. Pracownik kwalifikujący się do dodatku musi mieć 30+ lat oraz 3+ stażu pracy w firmie. Dodatek to dwukrotność pensji.

Funkcje - wywołania

```
-- Wylicz dodatek stażowy dla pracownika 104.
SELECT calculate_seniority_bonus(104) FROM   dual;
/
-- Wylicz dodatek stażowy dla wszystkich pracowników
SELECT e.*, calculate_seniority_bonus (employee_id)
FROM   employees e;
/
-- Pokaż maksymalne dodatki stażowe w departamentach. Pokaż liczbę pracowników departamentu.
SELECT d.name, count (employee_id)AS liczba, nvl(to_char(max(calculate_seniority_bonus(employee_id))), 'BRAK
BONUSU') AS max_bonus
FROM   employees e right join departments d USING (department_id)
GROUP  BY d.name
ORDER  BY 2 DESC;
/
-- Pokaż wysokości dodatków i liczbę pracowników, którzy go otrzymali. Wyłącz kandydatów i emerytów.
SELECT calculate_seniority_bonus(employee_id), count(*)
FROM   employees e join emp_status s USING (status_id)
WHERE  s.name NOT IN ( 'Kandydat', 'Emeryt' )
GROUP  BY calculate_seniority_bonus(employee_id)
ORDER  BY 1 DESC;
/
```

Funkcje - wywołania

```
-- wywołanie w bloku anonimowym
DECLARE
    c_emp_id NUMBER := 104;
    v_bonus  NUMBER;
BEGIN
    v_bonus := calculate_seniority_bonus (c_emp_id);
    dbms_output.put_line('Employee with id '|| c_emp_id || ' achieved bonus ' || v_bonus);
END;
/

-- wywołanie w innej funkcji
CREATE OR replace FUNCTION calculate_total_bonus(p_id NUMBER)
RETURN NUMBER
AS
  v_sen_bonus NUMBER;
  c_magic_bonus CONSTANT NUMBER := 1000;
BEGIN
    v_sen_bonus := calculate_seniority_bonus(p_id);

    RETURN v_sen_bonus + c_magic_bonus;
END;

/
```

Funkcje - ćwiczenia

1. Napisz funkcję, która wyliczy roczną wartość podatku pracownika. Zakładamy podatek progresywny. Początkowo stawka to 15%, po przekroczeniu progu 100000 stawka wynosi 25%.
2. Stwórz widok łączący departamenty, adresy i kraje. Napisz zapytanie, które pokaże sumę zapłaconych podatków w krajach.
3. Napisz funkcję, która wyliczy dodatek funkcyjny dla kierowników zespołów. Dodatek funkcyjny powinien wynosić 10% pensji za każdego podległego pracownika, ale nie może przekraczać 50% miesięcznej pensji.
4. Zmodyfikuj funkcję `calculate_total_bonus`, żeby wyliczała całość dodatku dla pracownika (stażowy i funkcyjny).

Procedury

- Nazwany blok PL/SQL.
- Może przyjmować parametry.
- Może zwracać wynik poprzez parametry wyjściowe.
- Na ogół procedura jest używana do zrealizowania skutku ubocznego (wypisanie na ekran, modyfikacja danych w tabeli).
- Procedura jest kompilowana i przechowywana w bazie.
- Wywołanie procedury:
  - ✅ z bloku anonimowego
  - ✅ z innej procedury i funkcji
  - ✅ polecenie interfejsu SQLPlus EXEC [UTE]
  - ⚠️ NIE z polecenia SQL

Parametry - tryby przekazywania

IN	OUT	IN OUT
Tryb domyślny	Musi być jawnie określony	Musi być jawnie określony
Wartość jest przesyłana do podprogramu	Wartość zwracana do środowiska wywołania	Wartość przesłana do podprogramu; oraz zwracana do środowiska wywołującego
Wartość stała, niezmienna *	Niezainicjowana zmienna (NULL) *	Zainicjowana zmienna *
Środowisko wywołania może przekazać stałą, literał, wyrażenia, lub zainicjowaną zmienną.	Musi być zmienną	Musi być zmienną

*) z perspektywy wnętrza procedury/funkcji

Procedury

```
CREATE [ OR REPLACE] PROCEDURE name [parameters] [(parameter1 [mode1]
datatype1, ...)]
IS | AS
  [local_variable_declarations;...]
BEGIN
  --actions;
EXCEPTION
  WHEN
  EXCEPTION
    --actions;
END [name];
```

Procedury - przykład

```
CREATE OR replace PROCEDURE add_candidate (p_name VARCHAR2, p_surname VARCHAR2, p_birth_date DATE,p_gender VARCHAR2,p_pos_name
VARCHAR2,p_dep_name VARCHAR2)
AS
  v_pos_id   NUMBER; v_dep_id   NUMBER; v_cand_num NUMBER;
  c_candidate_status CONSTANT NUMBER := 304;
  c_num_max CONSTANT NUMBER := 2;
BEGIN
    SELECT position_id INTO   v_pos_id  FROM   positions  WHERE  name = p_pos_name;
    SELECT department_id INTO   v_dep_id FROM   departments WHERE  name = p_dep_name;

    SELECT count(employee_id) INTO   v_cand_num
    FROM   employees
    WHERE  department_id = v_dep_id AND status_id = c_candidate_status;

    IF v_cand_num < c_num_max THEN
      INSERT INTO employees
       VALUES (NULL, p_name, p_surname, p_birth_date, p_gender, c_candidate_status, NULL, NULL, v_dep_id, v_pos_id, NULL);
      dbms_output.put_line ('Dodano kandydata '|| p_name|| ' '|| p_surname);
    ELSE
      dbms_output.put_line ('Za duzo kandydatów w departamencie: '|| p_dep_name);
    END IF;

EXCEPTION
  WHEN no_data_found THEN
            dbms_output.put_line ('Niepoprawna nazwa stanowiska i/lub zakładu');
            RAISE;
  WHEN too_many_rows THEN
            dbms_output.put_line ('Nieunikalna nazwa stanowiska i/lub zakładu');
            RAISE;
END;
/
```

Procedura dodająca kandydatów do departamentów. W departamencie nie może być > 2 kandydatów. Parametry to imię, nazwisko, data urodzenia, płeć, nazwa stanowiska i nazwa departamentu.

Procedury - wywołanie

```
-- wykonanie procedury
exec add_candidate ('Jan', 'Janowski', SYSDATE, 'M', 'Programista', 'Produkcja');
/
-- blok anonimowy
BEGIN
    add_candidate ('Jan', 'Janowski', SYSDATE, 'M', 'Programista', 'Produkcja');
END;
/
-- procedurę można wywołać z innej procedury
CREATE OR replace PROCEDURE load_data
AS
BEGIN
    add_candidate ('Jan', 'Janowski', SYSDATE, 'M', 'Programista', 'Produkcja');
END;
/
-- procedurę można wywołać z funkcji
CREATE OR replace FUNCTION load_data_fun
RETURN NUMBER
AS
  v_number NUMBER;
BEGIN
    add_candidate ('Jan', 'Janowski', SYSDATE, 'M', 'Programista', 'Helpdesk');

    SELECT count(*)INTO   v_number FROM   employees
    WHERE  name = 'Jan' AND surname = 'Janowski';
    RETURN v_number;
END;
```

Procedury - ćwiczenia

1. Napisz procedurę, która wykona zmianę stanowiska pracownika. Procedura powinna przyjmować identyfikator pracownika oraz identyfikator jego nowego stanowiska.
2. Sprawdź działanie procedury wywołując ją z bloku anonimowego.
3. Napisz procedurę, która zdegraduje zespołowego kierownika o danym identyfikatorze. Na nowego kierownika zespołu powołaj najstarszego z jego dotychczasowych podwładnych.
4. Sprawdź działanie procedury.

Praca domowa

1. Napisz funkcję, która będzie tworzyła bazowy login dla każdego pracownika. Login ma się składać z pierwszej litery imienia i maksymalnie 7 znaków z nazwiska.
2. Napisz procedurę, która będzie zapisywać login pracownika do nowej kolumny w tabeli employees (dodaj ją). Zadbaj o to, żeby zapisywany login był unikalny (np. poprzez dodanie numerów do bazowego loginu).
3. Sprawdź działanie trybów przekazania parametrów do procedury (IN, IN OUT i OUT).