



**Wydział Elektroniki
i Technik Informatycznych**

POLITECHNIKA WARSZAWSKA

Bazy Danych 1

edycja 21L

Laboratorium 9

Przebieg laboratorium

 Pakiety

 Wyzwalacze

 Kursory

 Dynamiczny SQL*



[PACKAGES - Oracle docs](#)



[TRIGGERS - Oracle docs](#)



[CURSORS - Oracle docs](#)

Pakiety

- Pakiet to obiekt bazodanowy, będący kontenerem pozwalającym na logiczne grupowanie podprogramów PL/SQL, kursorów, wyjątków...
- Pakiet składa się ze
 - specyfikacji (deklaracji dostępnych funkcji, procedur)
 - ciała (kod wykonywalny)
- Przykładowe pakiety bazodanowe: DBMS_OUTPUT, DBMS_RANDOM, DBMS_UTILITY, SDO_GEOM, UTL_HTTP

Pakiety

```
CREATE [OR REPLACE] PACKAGE package_name
IS | AS
public type and variable declarations
public subprogram specifications
END [package_name];
```

Definicja specyfikacji pakietu

```
CREATE [OR REPLACE] PACKAGE BODY package_name
IS | AS
private type and variable declarations
subprogram bodies
[BEGIN initialization statements]
END [package_name];
```

Definicja ciała pakietu

Pakiety - przykład

```
CREATE OR REPLACE PACKAGE emp_management  
AS
```

```
    FUNCTION calculate_seniority_bonus (p_id NUMBER) RETURN NUMBER;
```

```
    PROCEDURE add_candidate (p_name VARCHAR2, p_surname VARCHAR2, p_birth_date DATE, p_gender  
        VARCHAR2, p_pos_name VARCHAR2, p_dep_name VARCHAR2);
```

```
END;
```

```
/
```

```
CREATE OR REPLACE PACKAGE BODY emp_management  
AS
```

```
    FUNCTION calculate_seniority_bonus(p_id NUMBER) RETURN NUMBER
```

```
    AS
```

```
        /* kod funkcji calculate_seniority_bonus */
```

```
    END calculate_seniority_bonus;
```

```
    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
```

```
        /* kod procedury add_candidate */
```

```
    END add_candidate;
```

```
END emp_management;
```

Pakiety - ćwiczenia

1. Uzupełnij ciało pakietu z poprzedniego slajdu za pomocą definicji funkcji `calculate_seniority_bonus` oraz procedury `add_candidate`, które pojawiły się na poprzednich zajęciach. Następnie wywołaj te podprogramy z wykorzystaniem nazwy pakietu.
2. Dodaj do pakietu prywatną funkcję `create_base_login`, która będzie generowała bazowy login pracownika (ćwiczenie z pracy domowej BD1_8). Sprawdź możliwość wywołania tej funkcji.

Wyzwalacze

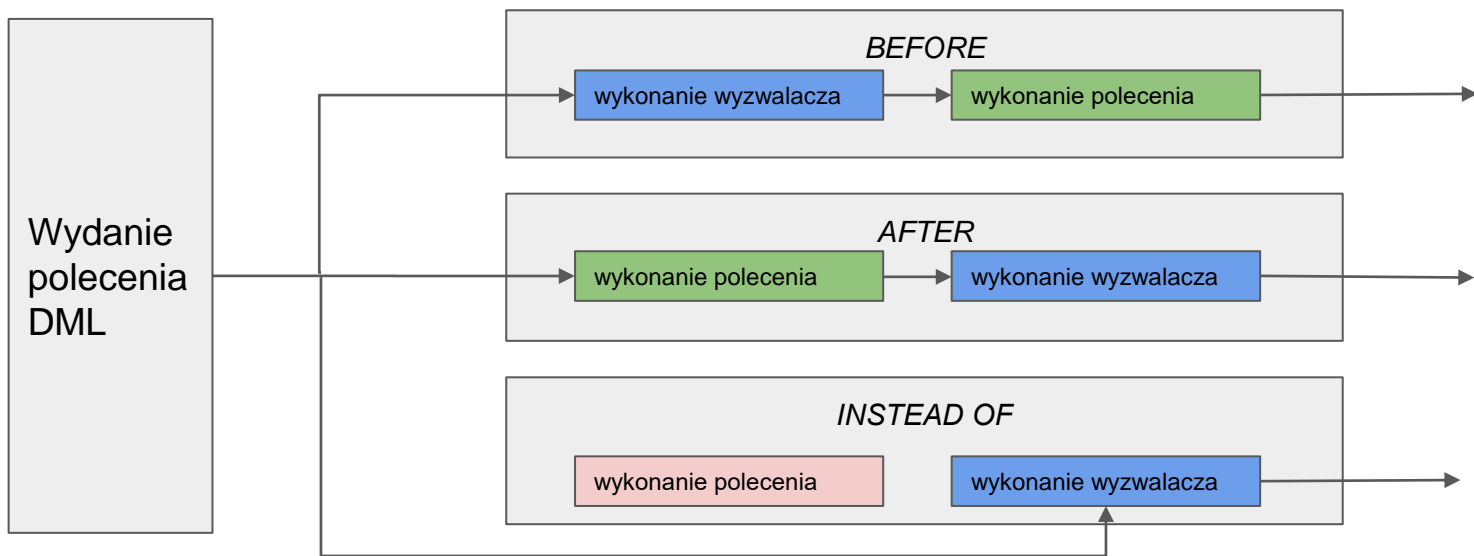
- Wyzwalacze (ang. *triggers*) są obiektami języka PL/SQL definiowanymi przez użytkownika i automatycznie uruchamianymi po wystąpieniu odpowiednich zdarzeń w bazie danych.
- Przykładowymi zdarzeniami w bazie danych, które mogą uruchomić wyzwalacz są np.:
 - definiowanie nowych obiektów BD (DDL triggers)
 - polecenia DML wykonywane na tabelach (DML triggers)
 - zdarzenia systemowe BD np. logowanie lub wylogowanie użytkownika BD, zamknięcie BD, uruchomienie BD itp (system triggers)

Wyzwalacze dla poleceń DML - charakterystyka

- wyzwalacz może być uruchamiany jednokrotnie (*statement-level trigger*) albo dla każdego modyfikowanego wiersza tabeli (widoku) osobno (*row-level trigger*).
- wyzwalacz może być uruchamiany przed lub po wykonaniu polecenia DML.
- możliwe jest określenie DML, dla którego będzie uruchamiany wyzwalacz (INSERT, UPDATE, DELETE) oraz łączenie różnych typów poleceń DML w jednym wyzwalaczu.
- możliwe jest definiowanie wyzwalaczy INSTEAD OF, które zamiast polecenia DML wykonują inne określone działania.
- możliwe jest ograniczenie uruchamiania wyzwalacza za pomocą klauzuli WHEN. Wyzwalacz będzie uruchamiany tylko, jeśli spełnione są warunki określone w klauzuli WHEN.

Uruchomienie wyzwalacza - BEFORE, AFTER, INSTEAD OF

- Wyzwalacze DML mogą być uruchamiane przed (BEFORE), po (AFTER) lub zamiast (INSTEAD OF) polecenia DML.



Wyzwalacze DML - składnia (1)

Składnia polecenia tworzącego wyzwalacz DML jest następująca:

```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza_DML
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] DELETE [OR] UPDATE [OF lista_kolumn]} ON nazwa_tabeli
[FOR EACH ROW]
[WHEN (... predykaty logiczne ...)]
[DECLARE]
    .. deklaracje i definicje zmiennych ..
BEGIN
    .. instrukcje języka PL/SQL ..
[EXCEPTION]
    .. obsługa wyjątków ..
END [nazwa_wyzwalacza_DML];
```

Wyzwalacze DML - składnia (2)

```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza_DML
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] DELETE [OR] UPDATE [OF lista_kolumn]} ON nazwa_tabeli
[FOR EACH ROW]
[WHEN (.. predykaty logiczne ..)]
[DECLARE]
    .. deklaracje i definicje zmiennych ..
BEGIN
    .. instrukcje języka PL/SQL ..
[EXCEPTION]
    .. obsługa wyjątków ..
END [nazwa_wyzwalacza_DML];
```

Określenie nazwy wyzwalacza

```
graph LR; A[Określenie nazwy wyzwalacza] --> B[nazwa_wyzwalacza_DML]; A --> C[nazwa_wyzwalacza_DML];
```

Wyzwalacze DML - składnia (3)

```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza_DML
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] DELETE [OR] UPDATE [OF lista_kolumn]} ON nazwa_tabeli
[FOR EACH ROW]
[WHEN (.. predykaty logiczne ..)]
[DECLARE]
    .. deklaracje i definicje zmiennych ..
BEGIN
    .. instrukcje języka PL/SQL ..
[EXCEPTION]
    .. obsługa wyjątków ..
END [nazwa_wyzwalacza_DML];
```

Określenie momentu uruchomienia wyzwalacza (wybranie **jednej** z opcji jest obowiązkowe):

- BEFORE przed poleceniem DML
- AFTER po poleceniu DML
- INSTEAD OF - zamiast polecenia DML (nie dla tabel)

Wyzwalacze DML - składnia (4)


```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza_DML
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] DELETE [OR] UPDATE [OF lista_kolumn]} ON nazwa_tabeli
[FOR EACH ROW]
[WHEN (.. predykaty logiczne ..)]
[DECLARE]
    .. deklaracje i definicje zmiennych ..
BEGIN
    .. instrukcje języka PL/SQL ..
[EXCEPTION]
    .. obsługa wyjątków ..
END [nazwa_wyzwalacza_DML];
```

Określenie dla jakiego (jakich) poleceń DML uruchomiony ma być wyzwalacz:

- można wskazać więcej niż jedno polecenie oddzielając słowa kluczowe spójnikiem OR.
- w przypadku polecenia UPDATE możliwe jest wskazanie listy kolumn, których ma dotyczyć polecenie UPDATE. Kolumny oddzielamy przecinkiem. Wyzwalacz będzie wywołany tylko jeśli polecenie UPDATE dotyczy którejś z kolumn z listy.
- wybranie **przynajmniej jednej** opcji jest obowiązkowe.

Wyzwalacze DML - składnia (5)


```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza_DML
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] DELETE [OR] UPDATE [OF lista_kolumn]} ON nazwa_tabeli
[FOR EACH ROW]
[WHEN (.. predykaty logiczne ..)]
[DECLARE]
    .. deklaracje i definicje zmiennych ..
BEGIN
    .. instrukcje języka PL/SQL ..
[EXCEPTION]
    .. obsługa wyjątków ..
END [nazwa_wyzwalacza_DML];
```



- Wskazanie, czy wyzwalacz ma być uruchamiany dla każdego modyfikowanego wiersza (dodajemy opcję FOR EACH ROW), czy też jednokrotnie dla polecenia (pomijamy tę opcję).
- Np. w przypadku wyzwalacza dla polecenia UPDATE oraz dla każdego wiersza (FOR EACH ROW), wyzwalacz ten będzie uruchamiany dla każdego wiersza, który modyfikowany jest przez polecenie UPDATE.

Wyzwalacze DML - składnia (6)

```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza_DML
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] DELETE [OR] UPDATE [OF lista_kolumn]} ON nazwa_tabeli
[FOR EACH ROW]
[WHEN (.. predykaty logiczne ..)]
[DECLARE]
    .. deklaracje i definicje zmiennych ..
BEGIN
    .. instrukcje języka PL/SQL ..
[EXCEPTION]
    .. obsługa wyjątków ..
END [nazwa_wyzwalacza_DML];
```



Określenie warunków, dla których wyzwalacz ma być uruchomiony np. jeśli modyfikowana kolumna, dodawany jest nowy wiersz danych o specyficznych wartościach kolumn itp. Klauzula WHEN jest opcjonalna.

Wyzwalacze DML - składnia (7)

```
CREATE [OR REPLACE] TRIGGER nazwa_wyzwalacza_DML
{BEFORE | AFTER | INSTEAD OF}
{INSERT [OR] DELETE [OR] UPDATE [OF lista_kolumn]} ON nazwa_tabeli
[FOR EACH ROW]
[WHEN (... predykaty logiczne ...)]
[DECLARE]
    .. deklaracje i definicje zmiennych ..
BEGIN
    .. instrukcje języka PL/SQL ..
[EXCEPTION]
    .. obsługa wyjątków ..
END [nazwa_wyzwalacza_DML];
```

Blok języka PL/SQL

Wyzwalacze DML - FOR EACH ROW - uwagi

- W przypadku wyzwalaczy FOR EACH ROW możliwe jest wykorzystanie dwóch pseudozmiennych o nazwach *old* i *new*.
- Pseudozmienne te pozwalają odwołać się do *starych* i *nowych* wartości modyfikowanych wierszy używając nazw kolumn ze zmienianej tabeli.
- Pseudozmienne *old* i *new*:
 - **nie są poprzedzone** znakiem **:** jeśli pojawiają się w klauzuli WHEN wyzwalacza;
 - **są poprzedzone** znakiem **:** jeśli pojawiają się w ramach bloku PL/SQL wyzwalacza.

Pseudozmienne *old* i *new* - użycie

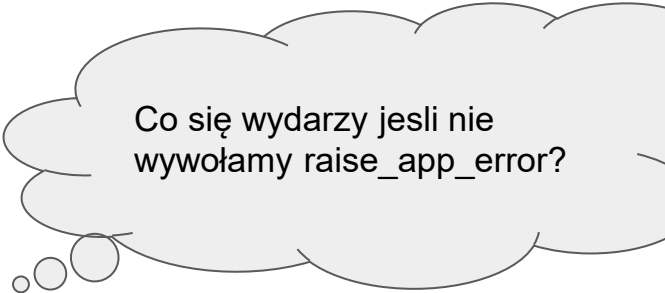
- Przy zapisywaniu do pseudozmiennych nowych wartości:
 - dla wyzwalaczy typu BEFORE **możliwe jest** przypisywanie nowych wartości do pseudozmiennej *new*, ale nie do *old*.
 - dla wyzwalaczy typu AFTER oraz INSTEAD OF **nie jest możliwe** przypisywanie nowych wartości do pseudozmiennych *new* i *old*.
- Możliwe jest odczytywanie pseudozmiennych:
 - dla wyzwalaczy INSERT pseudozmiennej *new*.
 - dla wyzwalaczy DELETE pseudozmiennej *old*.
 - dla wyzwalaczy UPDATE pseudozmiennych *new* oraz *old*.

Wyzwalacze - przykłady (1)

Przykład: Wyzwalacz `tg_salary_emp` wyświetla błąd, jeśli zarobki nowych pracowników (lub modyfikacja zarobków dla aktualnych pracowników) nie mieszczą się w widełkach płacowych dla stanowiska zajmowanego przez pracownika. Próba dodania pracownika o nieodpowiednich zarobkach powinna zostać powstrzymana.

```
CREATE OR REPLACE TRIGGER tg_salary_emp
BEFORE INSERT or UPDATE ON employees FOR EACH ROW
DECLARE
    v_min_sal positions.min_salary%TYPE;
    v_max_sal positions.max_salary%TYPE;
BEGIN
    SELECT min_salary, max_salary INTO v_min_sal, v_max_sal
    FROM positions WHERE position_id = :new.position_id;

    IF :new.salary NOT BETWEEN v_min_sal AND v_max_sal THEN
        dbms_output.put_line('Zarobki pracownika spoza zakresu płac: ' || v_min_sal || ' ' || v_max_sal);
        raise_application_error(-20001, 'Przekroczony zakres płacy');
    END IF;
END;
```



Co się wydarzy jeśli nie
wywołamy `raise_app_error`?

Wyzwalacze - przykłady (2)

Przykład: Wyzwalacz *tg_emp_ph* dla każdej zmiany stanowiska danego pracownika zapisze poprzednie stanowisko w tabeli *positions_history*.

```
CREATE OR REPLACE TRIGGER tg_emp_ph
AFTER UPDATE OF position_id ON employees FOR EACH ROW
WHEN (new.position_id != old.position_id)
DECLARE
    v_date_start DATE ;
BEGIN
    SELECT MAX(date_end) INTO v_date_start FROM positions_history where employee_id=:old.employee_id;

    IF v_date_start IS NULL THEN
        v_date_start := :old.date_employed;
    END IF;

    INSERT INTO positions_history (employee_id, position_id, date_start, date_end)
    VALUES (:old.employee_id, :old.position_id, v_date_start, SYSDATE);

END;
```

Trigger wstawia rekordy do tabeli. Co się wydarzy w przypadku ROLLBACK?

Wyzwalacze - przydatne polecenia

```
-- wyłącz wyzwalacz tg_emp_ph (nie usuwa wyzwalacza)
```

```
ALTER TRIGGER tg_emp_ph DISABLE;
```

```
-- włącz wyzwalacz tg_emp_ph
```

```
ALTER TRIGGER tg_emp_ph ENABLE;
```

```
-- usuń wyzwalacz tg_emp_ph
```

```
DROP TRIGGER tg_emp_ph;
```

```
-- pokaż wszystkie wyzwalacze na tabeli employees
```

```
SELECT * FROM user_triggers WHERE table_name = 'EMPLOYEES';
```

```
-- wyłącz wszystkie wyzwalacze na tabeli employees
```

```
ALTER TABLE employees DISABLE ALL TRIGGERS;
```

```
-- włącz wszystkie wyzwalacze na tabeli employees
```

```
ALTER TABLE employees ENABLE ALL TRIGGERS;
```

Wyzwalacze - ćwiczenia

1. Stwórz wyzwalacz, który podczas uaktualniania zarobków pracownika wyświetli podatek 20% procent od nowych zarobków. Przetestuj działanie.
2. Stwórz wyzwalacz, który po dodaniu nowego pracownika, usunięciu pracownika lub modyfikacji zarobków pracowników wyświetli aktualne średnie zarobki wszystkich pracowników. Przetestuj działanie.
3. Stwórz wyzwalacz, który dla każdego nowego pracownika nieposiadającego managera, ale zatrudnionego w departamencie, przypisze temu pracownikowi managera będącego jednocześnie managerem departamentu, w którym ten pracownik pracuje. **Wykorzystaj** klauzulę WHEN wyzwalacza. Przetestuj działanie.
4. Rozwiąż ponownie ćwiczenie nr 4, ale tym razem **nie wykorzystuj** klauzuli WHEN wyzwalacza. Przetestuj działanie.
5. Stwórz wyzwalacz który będzie weryfikował, że w firmie pracuje tylko jeden Prezes.

⚡ Wyzwalacze - dyskusja ⚡

When are triggers dangerous?

Triggers are like Pringles: Once you pop, you can't stop. One of the greatest challenges for architects and developers is to ensure that triggers are used only as needed, and to not allow them to become a one-size-fits-all solution for any data needs that happen to come along. Adding TSQL to triggers is often seen as faster and easier than adding code to an application, but the cost of doing so is compounded over time with each added line of code.

<https://www.red-gate.com/simple-talk/sql/database-administration/sql-server-triggers-good-scary/>



Wyzwalacze - dyskusja



Advantages

Disadvantages

1. SQL Trigger provides an alternative way to check data integrity.
2. SQL trigger can catch the errors in business logic in the database level.
3. SQL trigger provides an alternative way to run scheduled tasks. With SQL trigger, you don't have to wait to run the scheduled tasks. You can handle those tasks before or after changes being made to database tables.
4. SQL trigger is very useful when you use it to audit the changes of data in a database table.

1. Even though we said that SQL triggers can provide extended validations, it can not be used to replace all other validations. Some simple validations can be done in the application level.
2. SQL Triggers executes invisibly from client-application which connects to the database server **so it is difficult to figure out what happen underlying database layer.**
3. SQL Triggers run every updates made to the table therefore **it adds workload to the database and cause system runs slower.**

<https://medium.com/@charithra/database-trigger-guards-at-a-ticking-point-c2d7e229564>

Kursory

- Kursory są strukturami języka PL/SQL pozwalającymi na dostęp do poszczególnych wierszy rezultatu zapytania, a następnie ich przetwarzanie.
- Kursory mogą być jawne lub niejawne. Niejawne kursory są na ogół wykorzystywane w pętli FOR, kursory jawne trzeba zadeklarować w sekcji DECLARE.

Kursory niejawne

- Nie są deklarowane w sekcji DECLARE.
- Są automatycznie zarządzane przez bazę danych (nie ma potrzeby ich otwierania, zamykania i bezpośredniego ładowania wierszy danych za ich pomocą).
- Mogą zostać wykorzystane w składni pętli FOR.
- Są automatycznie tworzone dla poleceń DML oraz zapytań.

Kursory niejawne - przykład

Przykład: Wykorzystując kursor niejawny oraz pętlę FOR wyświetli imię i nazwisko pracownika, a także nazwę jego stanowiska oraz nazwę departamentu w którym pracuje.

```
BEGIN
FOR r_emp IN (SELECT e.name as name, surname, p.name as position,
                  d.name as department
               FROM employees e JOIN
                  positions p USING (position_id) JOIN
                  departments d USING (department_id))

    dbms_output.put_line('Dane prac.: ' || r_emp.name || ' ' || r_emp.surname
                        || ' ' || r_emp.position || ' ' || r_emp.department);
END LOOP;
END;
```

Niejawnie zadeklarowany rekord przechowujący jeden wiersz danych kursora

Zapytanie definiujące kursor niejawny

Kursory jawne

- Kursory stosowane są w programach PL/SQL (bloki anonimowe, nazwane).
- Kursory (jak zmienne) definiujemy w sekcji DECLARE.
- Kursory mogą posiadać parametry.
- Cursor powinien być zdefiniowany w oparciu o zapytanie.
- Możliwe jest przeglądanie rezultatów zapisanych w kursorze za pomocą operacji FETCH.
- Cursor jawny musi zostać otwarty poleceniem OPEN przed pierwszym pobraniem wiersza za pomocą operacji FETCH. Niezamknięcie kursora poleceniem CLOSE powoduje wyciek pamięci (można też skorzystać z konstrukcji FOR).

Kursory jawne - przykład

- Zdefiniuj kursor przechowujący dane pracowników. Następnie wykorzystaj zdefiniowany kursor do wyświetlenia na ekranie podatku od zarobków dla pracowników (20% zarobków).

```
DECLARE
CURSOR cr IS
    SELECT * FROM employees;
v_rec_employees employees%ROWTYPE;

BEGIN
    OPEN cr;
    LOOP
        EXIT WHEN cr%NOTFOUND;
        FETCH cr INTO v_rec_employees;
        dbms_output.put_line('Podstawowe dane pracownika: ' || v_rec_employees.name || ' ' ||
            v_rec_employees.surname || ' ' || v_rec_employees.salary || ' ' ||
            v_rec_employees.salary || ' ' || 'Podatek: ' || 0.2*v_rec_employees.salary);
    END LOOP;
    CLOSE cr;
END;
```

Deklaracja kursora

Otwarcie kursora

Warunek zakończenia pętli

Załadowanie wiersza danych z kursora

Zamknięcie kursora

Kursory - uwaga

- Kursory (jawne i niejawne) przedstawione w poprzednich slajdach to kursory udostępniające dane do odczytu
- Kursory umożliwiające modyfikację danych są poza zakresem zajęć

Kursory - ćwiczenia

1. Przygotuj procedurę PL/SQL, która z wykorzystaniem jawnego kursora udostępni średnie zarobki dla każdego z departamentów. Następnie wykorzystując ten cursor wyświetl imiona, nazwiska i zarobki pracowników, którzy zarabiają więcej niż średnie zarobki w ich departamentach.
2. Przygotuj procedurę PL/SQL, która z wykorzystaniem jawnego kursora wyświetli *p_no_dept* departamenty największych budżetach, gdzie *p_no_dept* to parametr wejściowy procedury. Następnie wyświetl dane kierowników tych departamentów.
3. Wykorzystując niejawny cursor oraz deklaracje zmiennych/stałych podnieś o 2% pensje wszystkim pracownikom zatrudnionym w przeszłości (tzn. przed aktualnym stanowiskiem pracy) na co najmniej jednym stanowisku pracy.

✱ Dynamiczny SQL ✱

- Polecenia SQL (lub PL/SQL) mogą być dynamicznie uruchamiane z poziomu języka PL/SQL bez ich wcześniejszej kompilacji.
- Dynamiczny SQL może być szczególnie przydatny w sytuacji, kiedy na etapie kompilacji programu PL/SQL (np. procedury, pakietu) nie jest dokładnie znana np. treść zapytania SELECT, polecenia DML, lub polecenia DDL.
- Do dynamicznego wywołania polecenia możemy użyć instrukcji EXECUTE IMMEDIATE :

DECLARE

```
v_state VARCHAR2(100) := 'SELECT name FROM employees WHERE employee_id = 102';  
v_ename employees.name%TYPE;
```

BEGIN

```
EXECUTE IMMEDIATE v_state INTO v_ename;  
dbms_output.put_line('Employee name: ' || v_ename);
```

END;

⚡ Następne zajęcia - sprawdzian ⚡

Zakres:

- język PL/SQL (*)
 - bloki anonimowe
 - funkcje
 - procedury
 - pakiety
 - wyzwalacze
 - kursory

*) Na potrzeby testowania potrzebne będą także polecenia z grupy DML i DDL.

Praca domowa

1. Stwórz widok udostępniający dane pracowników (id, imię, nazwisko, data urodzenia, zarobki) oraz dane stanowisk (id, nazwa). Następnie stwórz wyzwalacz typu INSTEAD OF dla tego widoku, który po wykonaniu operacji INSERT dla widoku doda nowego pracownika oraz (jeśli potrzeba) stanowisko do tabel bazowych employees oraz positions.
2. Rozwiąż niezrealizowane ćwiczenia z poprzednich slajdów.
3. Czy jedno zdarzenie może uruchomić kilka wyzwalaczy? Jeśli tak, to w jakiej kolejności zostaną wykonane?
4. Jakie są negatywne skutki użycia wyzwalaczy?
5. Kiedy warto używać wyzwalaczy?
6. Stwórz tabelę projects_history a następnie zrealizuj wyzwalacz, który będzie logował każdą zmianę (tylko update) w tabeli projects. Zapisz starą i nową wartość każdej kolumny.