# Bazy Danych - Laboratorium 9

Student:

```
Bartłomiej Krawczyk
```

Numer Albumu:

```
310774
```

```
-- 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.

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
        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 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
        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 add_candidate;

    FUNCTION create_base_login(p_id NUMBER)
    RETURN VARCHAR2
    AS
        v_login VARCHAR2(8 CHAR);
    BEGIN
        SELECT SUBSTR(name, 1, 1) || SUBSTR(surname, 1, 7)
        INTO v_login
        FROM employees
        WHERE employee_id = p_id;
```

```
        RETURN v_login;
    END create_base_login;
END emp_management;
/

SELECT emp_management.calculate_seniority_bonus(employee_id)
FROM employees;

EXEC emp_management.add_candidate('test', 'test', '01/01/01', 'K', 'Kadrowy',
'Administracja');

ROLLBACK;

-- 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.

EXEC emp_management.create_base_login(125);  -- Nie jest zadeklarowana

--------------------------------------------------------------------------------


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');
        IF :new.salary < v_min_sal THEN
            :new.salary := v_min_sal;
        ELSE
            :new.salary := v_max_sal;
        END IF;
    END IF;
END;
/

UPDATE employees SET salary = 10000 WHERE employee_id = 125;

SELECT salary FROM employees WHERE employee_id = 125;

ROLLBACK;


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;
/

SELECT * FROM positions_history WHERE employee_id = 118;

UPDATE employees set position_id = 105 WHERE employee_id = 118;

SELECT * FROM positions_history WHERE employee_id = 118;

--------------------------------------------------------------------------------

-- 1. Stwórz wyzwalacz, który podczas uaktualniania zarobków pracownika wyświetli
podatek 20% procent od nowych zarobków. Przetestuj działanie.

CREATE OR REPLACE TRIGGER tg_tax_emp
AFTER UPDATE OF salary ON employees FOR EACH ROW
DECLARE
    v_tax NUMBER;
    c_tax  CONSTANT NUMBER := 0.2;
BEGIN

    v_tax := :new.salary * c_tax;

    dbms_output.put_line('Podatek od nowych zarobków: ' || v_tax);
END;
/

UPDATE employees
SET salary = 8000
WHERE employee_id = 125;

-- 2. Stwórz wyzwalacz, który po dodaniu nowego pracownika, usunięciu pracownika
lub modyfikacji zarobków pracowników wyświetli aktualne średnie zarobki wszystkich
pracowników. Przetestuj działanie.

CREATE OR REPLACE TRIGGER tg_avg_emp
AFTER INSERT OR DELETE OR UPDATE OF salary ON employees
DECLARE
    v_avg_salary NUMBER;
BEGIN
    SELECT AVG(salary) INTO v_avg_salary
    FROM employees;
```

```
    dbms_output.put_line('Nowa średnia zarobków: ' || v_avg_salary);
END;
/

UPDATE employees
SET salary = 8001
WHERE employee_id = 125;


-- 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.

CREATE OR REPLACE TRIGGER tg_manager_emp
BEFORE INSERT ON employees
FOR EACH ROW
WHEN (new.manager_id IS NULL AND new.department_id IS NOT NULL)
DECLARE
    v_manager_id NUMBER;
BEGIN
    SELECT manager_id INTO v_manager_id
    FROM departments
    WHERE department_id = :new.department_id;

    :new.manager_id := v_manager_id;
END;
/

SELECT * FROM employees WHERE manager_id IS NULL AND department_id IS NOT NULL
FETCH FIRST 1 ROW ONLY;

DECLARE
    v_emp employees%ROWTYPE;
BEGIN
    SELECT * INTO v_emp FROM (SELECT * FROM employees WHERE manager_id IS NULL AND
department_id IS NOT NULL FETCH FIRST 1 ROW ONLY);

    INSERT INTO employees
    VALUES (NULL, v_emp.name, v_emp.surname, v_emp.birth_date, v_emp.gender,
v_emp.status_id, v_emp.salary, v_emp.date_employed, v_emp.department_id,
v_emp.manager_id, v_emp.position_id);
END;
/

SELECT * FROM employees ORDER BY employee_id DESC FETCH FIRST 1 ROW ONLY;

-- 4. Rozwiąż ponownie ćwiczenie nr 3, ale tym razem nie wykorzystuj klauzuli WHEN
wyzwalacza. Przetestuj działanie.

CREATE OR REPLACE TRIGGER tg_manager_emp
BEFORE INSERT ON employees
FOR EACH ROW
```

```
DECLARE
    v_manager_id NUMBER;
BEGIN
    IF (:new.manager_id IS NULL AND :new.department_id IS NOT NULL) THEN
        SELECT manager_id INTO v_manager_id
        FROM departments
        WHERE department_id = :new.department_id;

        :new.manager_id := v_manager_id;
    END IF;
END;
/

SELECT * FROM employees WHERE manager_id IS NULL AND department_id IS NOT NULL
FETCH FIRST 1 ROW ONLY;

DECLARE
    v_emp employees%ROWTYPE;
BEGIN
    SELECT * INTO v_emp FROM (SELECT * FROM employees WHERE manager_id IS NULL AND
department_id IS NOT NULL FETCH FIRST 1 ROW ONLY);

    INSERT INTO employees
    VALUES (NULL, v_emp.name, v_emp.surname, v_emp.birth_date, v_emp.gender,
v_emp.status_id, v_emp.salary, v_emp.date_employed, v_emp.department_id,
v_emp.manager_id, v_emp.position_id);
END;
/

SELECT * FROM employees ORDER BY employee_id DESC FETCH FIRST 1 ROW ONLY;

-- 5. Stwórz wyzwalacz który będzie weryfikował, że w firmie pracuje tylko jeden
Prezes.

INSERT INTO positions
VALUES (118, 'Prezes', 0, 99999);

CREATE OR REPLACE TRIGGER tg_one_boss
BEFORE INSERT OR UPDATE of position_id ON employees
DECLARE
    v_position_id NUMBER;
    v_count NUMBER;
BEGIN
    SELECT position_id
    INTO v_position_id
    FROM positions WHERE name Like 'Prezes';

    SELECT count(employee_id)
    INTO v_count
    FROM employees
    WHERE position_id = v_position_id;

    IF v_count >= 1 THEN RAISE_APPLICATION_ERROR(-20001, 'Wieciej niz jeden
prezes'); END IF;
```

```
END;
/

UPDATE employees SET position_id = 118 WHERE employee_id = 101;
UPDATE employees SET position_id = 118 WHERE employee_id = 102;
UPDATE employees SET position_id = 118 WHERE employee_id = 103;

--------------------------------------------------------------------------------

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

DECLARE
    CURSOR cr IS
        SELECT * FROM employees;
    v_rec_employees employees%ROWTYPE;
BEGIN
  OPEN cr;
  LOOP
    FETCH cr INTO v_rec_employees;
    EXIT WHEN cr%NOTFOUND;
    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;
/
--------------------------------------------------------------------------------

-- 1. Przygotuj procedurę PL/SQL, która z wykorzystaniem jawnego kursora udostępni
średnie zarobki dla każdego z departamentów. Następnie wykorzystując ten kursor
wyświetl imiona, nazwiska i zarobki pracowników, którzy zarabiają więcej niż
średnie zarobki w ich departamentach.
DECLARE
    CURSOR cr IS
        SELECT (SELECT AVG(e2.salary) FROM employees  e2 WHERE e2.department_id =
e1.department_id GROUP BY e2.department_id), name, surname, salary
        FROM employees e1;
    v_avg_sal NUMBER;
    v_name employees.name%TYPE;
    v_surname employees.surname%TYPE;
    v_salary employees.salary%TYPE;
BEGIN
```

```
    OPEN cr;
    LOOP
        FETCH cr INTO v_avg_sal, v_name, v_surname, v_salary;
        EXIT WHEN cr%NOTFOUND;

        IF v_salary > v_avg_sal THEN
            dbms_output.put_line ('Pracownik '|| v_name || ' '|| v_surname || ' '
|| v_salary);
        END IF;
    END LOOP;
    CLOSE cr;
END;
/

-- 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.

CREATE OR REPLACE PROCEDURE depr_max_budget (p_no_dept IN INTEGER)
AS
    CURSOR cr IS
        SELECT *
        FROM departments
        ORDER BY year_budget DESC
        FETCH FIRST p_no_dept ROWS ONLY;

v_depts departments%ROWTYPE;
v_manager employees%ROWTYPE;
BEGIN
    OPEN cr;
    LOOP
        FETCH cr INTO v_depts;
        EXIT WHEN cr%NOTFOUND;
        dbms_output.put_line(v_depts.name || ' ' || v_depts.year_budget);
    END LOOP;
    CLOSE cr;

    OPEN cr;
    LOOP
        FETCH cr INTO v_depts;
        EXIT WHEN cr%NOTFOUND;

        SELECT * INTO v_manager
        FROM employees
        WHERE employee_id = v_depts.manager_id;

        dbms_output.put_line(v_manager.name || ' ' || v_manager.surname);
    END LOOP;
    CLOSE cr;
END;
/
EXEC depr_max_budget(5);

-- 3. Wykorzystując niejawny kursor oraz deklaracje zmiennych/stałych podnieś o 2%
```

```
pensje wszystkim pracownikom zatrudnionym w przeszłości (tzn. przed aktualnym
stanowiskiem pracy) na co najmniej jednym stanowisku pracy.

SELECT * FROM employees e WHERE EXISTS(SELECT * FROM positions_history ph WHERE
ph.employee_id = e.employee_id) AND salary IS NOT NULL;

DECLARE
    c_rise CONSTANT NUMBER := 1.02;
BEGIN
    FOR v_emp IN (SELECT * FROM employees e WHERE EXISTS(SELECT * FROM
positions_history ph WHERE ph.employee_id = e.employee_id) AND salary IS NOT NULL)
    LOOP
        UPDATE employees
        SET salary = v_emp.salary * c_rise
        WHERE employee_id = v_emp.employee_id;
    END LOOP;
END;
/

SELECT * FROM employees e WHERE EXISTS(SELECT * FROM positions_history ph WHERE
ph.employee_id = e.employee_id) AND salary IS NOT NULL;

--------------------------------------------------------------------------------

-- 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.

CREATE OR REPLACE VIEW emp_view AS
    SELECT employee_id, e.name, surname, birth_date, salary, position_id, p.name
"position_name"
    FROM employees e
    JOIN positions p USING(position_id);

SELECT * FROM emp_view;

CREATE OR REPLACE TRIGGER tg_emp_view
INSTEAD OF INSERT ON emp_view
FOR EACH ROW
DECLARE
    v_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_count FROM positions WHERE position_id =
:new.position_id;
    IF v_count = 0 THEN
        INSERT INTO positions
        VALUES (:new.position_id, :new.name, NULL, NULL);  -- I cannot use aliases
- I don't know why
    END IF;
```

```
    INSERT INTO employees
    VALUES (:new.employee_id, :new.name, :new.surname, :new.birth_date, NULL,
NULL, :new.salary, NULL, NULL, NULL, :new.position_id);
END;
/

INSERT INTO emp_view
VALUES (1, 'Test', 'Testowy', '01/01/01', 10, 1, 'Test');


SELECT * FROM employees WHERE employee_id = 1;
SELECT * FROM positions WHERE position_id = 1;

-- 2. Rozwiąż niezrealizowane ćwiczenia z poprzednich slajdów.

-- Done

-- 3. Czy jedno zdarzenie może uruchomić kilka wyzwalaczy? Jeśli tak, to w jakiej
kolejności zostaną wykonane?

CREATE OR REPLACE TRIGGER tg_1_emp
AFTER UPDATE OF name ON employees FOR EACH ROW
BEGIN
    dbms_output.put_line('1');
END;
/


CREATE OR REPLACE TRIGGER tg_2_emp
AFTER UPDATE OF name ON employees FOR EACH ROW
BEGIN
    dbms_output.put_line('2');
END;
/

CREATE OR REPLACE TRIGGER tg_3_emp
AFTER UPDATE OF name ON employees FOR EACH ROW
BEGIN
    dbms_output.put_line('3');
END;
/

UPDATE employees
SET name = 'Test'
WHERE employee_id = 125;

-- 3
-- 2
-- 1

-- wyłącz wszystkie wyzwalacze na tabeli employees
ALTER TABLE employees DISABLE ALL TRIGGERS;
-- włącz wszystkie wyzwalacze na tabeli employees
ALTER TABLE employees ENABLE ALL TRIGGERS;
```

```
UPDATE employees
SET name = 'Test'
WHERE employee_id = 125;

-- 3
-- 2
-- 1

ALTER TRIGGER tg_2_emp DISABLE;
ALTER TRIGGER tg_2_emp ENABLE;

UPDATE employees
SET name = 'Test'
WHERE employee_id = 125;

-- 3
-- 2
-- 1

-- Tak może, w kolejności odwrotenej do kolejności kompilowania ich

-- 4. Jakie są negatywne skutki użycia wyzwalaczy?

-- Wyzwalacze są wykonywane niewidocznie dla aplikacji klienta -> klient może nie
wiedzieć czemu zmiana jenego parametru powoduje inne konsekwencje

-- 5. Kiedy warto używać wyzwalaczy?

-- Sprawdzając logikę biznesową na poziomie bazy danych - nie pozwolenie na
dodanie nieprawidłowych danych
-- Sprawdzając integralność danych

-- 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.

CREATE TABLE projects_history
(
      projects_history_id NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY START
WITH 1000  CONSTRAINT projects_history_pk PRIMARY KEY,
      old_project_id NUMBER,
      old_name VARCHAR2(50) NOT NULL,
      old_status NUMBER REFERENCES project_status(ps_id),
      old_owner NUMBER REFERENCES departments (department_id),
      old_estimated_budget NUMBER,
      old_used_budget NUMBER,
      old_date_start DATE,
      old_date_end DATE,
      new_project_id NUMBER,
      new_name VARCHAR2(50) NOT NULL,
      new_status NUMBER REFERENCES project_status(ps_id),
      new_owner NUMBER REFERENCES departments (department_id),
      new_estimated_budget NUMBER,
```

```
      new_used_budget NUMBER,
      new_date_start DATE,
      new_date_end DATE
);

CREATE OR REPLACE TRIGGER tg_log_proj
AFTER UPDATE ON projects FOR EACH ROW
BEGIN
    INSERT INTO projects_history
    VALUES (NULL,
            :old.project_id, :old.name, :old.status, :old.owner,
:old.estimated_budget, :old.used_budget, :old.date_start, :old.date_end,
            :new.project_id, :new.name, :new.status, :new.owner,
:new.estimated_budget, :new.used_budget, :new.date_start, :new.date_end);
END;
/

UPDATE projects
SET name = 'Test'
WHERE project_id = 100;

SELECT * FROM projects_history;


ROLLBACK;
```