Bazy i zbiory danych sprawozdanie

# Utworzenie struktury danych

**CREATE TABLE UZYTKOWNIK (**

**UZYTKOWNIK\_ID NUMBER GENERATED BY DEFAULT AS IDENTITY,**

**IMIE VARCHAR2(50) NOT NULL,**

**NAZWISKO VARCHAR2(50) NOT NULL,**

**PESEL VARCHAR2(11) NOT NULL,**

**DATA\_REJESTRACJI DATE DEFAULT SYSDATE,**

**EMAIL VARCHAR2(100) NOT NULL,**

**TELEFON VARCHAR2(15),**

**STAN\_KONTA NUMBER(10,2) DEFAULT 0,**

**PRIMARY KEY(UZYTKOWNIK\_ID)**

**);**

**CREATE TABLE ZAKLAD (**

**ZAKLAD\_ID NUMBER GENERATED BY DEFAULT AS IDENTITY,**

**NAZWA VARCHAR2(100) NOT NULL,**

**DYSCYPLINA VARCHAR2(50) NOT NULL,**

**DATA\_ZDARZENIA DATE NOT NULL,**

**KURS NUMBER(5,2) NOT NULL,**

**STATUS VARCHAR2(20) DEFAULT “Nierozstrzygnięty”,**

**PRIMARY KEY(ZAKLAD\_ID)**

**);**

**CREATE TABLE TRANSAKCJA (**

**TRANSAKCJA\_ID NUMBER GENERATED BY DEFAULT AS IDENTITY,**

**UZYTKOWNIK\_ID NUMBER NOT NULL,**

**RODZAJ VARCHAR2(20) NOT NULL,**

**KWOTA NUMBER(10,2) NOT NULL,**

**DATA\_TRANSAKCJI DATE DEFAULT SYSDATE,**

**ZAKLAD\_ID NUMBER,**

**PRIMARY KEY(TRANSAKCJA\_ID),**

**FOREIGN KEY(UZYTKOWNIK\_ID) REFERENCES UZYTKOWNIK(UZYTKOWNIK\_ID),**

**FOREIGN KEY(ZAKLAD\_ID) REFERENCES ZAKLAD(ZAKLAD\_ID)**

**);**

**CREATE TABLE ARCHIWUM (**

**ARCHIWUM\_ID NUMBER GENERATED BY DEFAULT AS IDENTITY,**

**NAZWA\_TABLICY VARCHAR2(50) NOT NULL,**

**ID\_W\_PIERWOTNEJ NUMBER NOT NULL,**

**DANE\_JSON CLOB,**

**DATA\_ARCHIWIZACJI DATE DEFAULT SYSDATE,**

**PRIMARY KEY(ARCHIWUM\_ID)**

**);**

**CREATE TABLE ARCHIWUM (**

**ARCHIWUM\_ID NUMBER GENERATED BY DEFAULT AS IDENTITY,**

**NAZWA\_TABLICY VARCHAR2(50) NOT NULL,**

**ID\_W\_PIERWOTNEJ NUMBER NOT NULL,**

**DANE\_JSON CLOB,**

**DATA\_ARCHIWIZACJI DATE DEFAULT SYSDATE,**

**PRIMARY KEY(ARCHIWUM\_ID)**

**);**

# Skrypt ładujący dane do bazy:

Stworzyliśmy skrypt, który ładuje dane do bazy danych w języku python z użyciem biblioteki cx\_Oracle.

import cx\_Oracle

import csv

import os

import re

# Funkcja do oczyszczania danych

def clean\_data(value):

return re.sub(r"[^\w\s.,-]", "", value.strip())

# Ścieżka do Instant Client

os.environ["PATH"] = r"C:\Users\M\Desktop\instant client\instantclient\_23\_6;" + os.environ["PATH"]

# Konfiguracja połączenia

dsn\_tns = cx\_Oracle.makedsn("213.184.8.44", 1521, service\_name="orcl")

connection = cx\_Oracle.connect(

user="usckim",

password="klaudiar",

dsn=dsn\_tns,

encoding="UTF-8"

)

# Ścieżka do pliku CSV

csv\_file\_path = "zaklad\_loader.csv"

try:

cursor = connection.cursor()

# Zapytanie SQL do wstawiania danych (bez ID)

insert\_sql = """

INSERT INTO ZAKLAD (

NAZWA,

DYSCYPLINA,

DATA\_ZDARZENIA,

KURS,

STATUS

)

VALUES (

:1,

:2,

TO\_DATE(:3, 'YYYY-MM-DD'),

:4,

:5

)

"""

with open(csv\_file\_path, "r", encoding="utf-8") as csv\_file:

reader = csv.reader(csv\_file, delimiter=",")

next(reader, None) # Pomijamy nagłówek

for row in reader:

# Oczyszczamy dane

row = [clean\_data(value) for value in row]

# Sprawdzamy liczbę kolumn

if len(row) != 5:

print(f"Pominięto wiersz z niepoprawną liczbą kolumn: {row}")

continue

# Przypisujemy wartości

nazwa, dyscyplina, data\_zdarzenia, kurs, status = row

kurs = kurs.replace(",", ".") # Zamiana przecinka na kropkę

# Walidacja liczby w kolumnie KURS

try:

kurs = float(kurs)

except ValueError:

print(f"Błędny format kursu: {kurs}. Pominięto rekord: {row}")

continue

# Wstawianie danych do bazy

try:

cursor.execute(insert\_sql, (

nazwa,

dyscyplina,

data\_zdarzenia,

kurs,

status

))

except cx\_Oracle.DatabaseError as e:

print(f"Błąd wstawiania rekordu: {row} - {e}")

continue

# Zatwierdzamy zmiany

connection.commit()

print("Dane zostały załadowane i zapisane w tabeli ZAKLAD.")

except cx\_Oracle.Error as e:

print("Wystąpił błąd podczas łączenia lub zapisu do bazy:", e)

finally:

if 'cursor' in locals():

cursor.close()

connection.close()

# Procedury, funkcje, wyzwalacze obsługujące bazę

### Dodawanie, usuwanie, aktualizacja rekordów

**Procedura tworzenia/edycji użytkownika:**

**CREATE OR REPLACE PROCEDURE STWORZ\_LUB\_EDYTUJ\_UZYTKOWNIKA (**

**p\_uzytkownik\_id IN NUMBER, -- może być NULL => nowy użytkownik**

**p\_imie IN VARCHAR2,**

**p\_nazwisko IN VARCHAR2,**

**p\_pesel IN VARCHAR2,**

**p\_email IN VARCHAR2,**

**p\_telefon IN VARCHAR2 DEFAULT NULL,**

**p\_stan\_konta IN NUMBER DEFAULT 0**

**)**

**IS**

**BEGIN**

**IF p\_uzytkownik\_id IS NULL THEN**

**-- Tworzymy nowego użytkownika**

**INSERT INTO UZYTKOWNIK (**

**IMIE,**

**NAZWISKO,**

**PESEL,**

**DATA\_REJESTRACJI,**

**EMAIL,**

**TELEFON,**

**STAN\_KONTA**

**)**

**VALUES (**

**p\_imie,**

**p\_nazwisko,**

**p\_pesel,**

**SYSDATE,**

**p\_email,**

**p\_telefon,**

**p\_stan\_konta**

**);**

**ELSE**

**-- Edytujemy istniejącego użytkownika**

**UPDATE UZYTKOWNIK**

**SET IMIE = p\_imie,**

**NAZWISKO = p\_nazwisko,**

**PESEL = p\_pesel,**

**EMAIL = p\_email,**

**TELEFON = p\_telefon,**

**STAN\_KONTA= p\_stan\_konta**

**WHERE UZYTKOWNIK\_ID = p\_uzytkownik\_id;**

**END IF;**

**END;**

**/**

Usuwanie użytkownika:

**CREATE OR REPLACE PROCEDURE USUN\_UZYTKOWNIKA(**

**p\_uzytkownik\_id IN NUMBER**

**)**

**IS**

**v\_imie UZYTKOWNIK.IMIE%TYPE;**

**v\_nazwisko UZYTKOWNIK.NAZWISKO%TYPE;**

**v\_pesel UZYTKOWNIK.PESEL%TYPE;**

**v\_data\_rej UZYTKOWNIK.DATA\_REJESTRACJI%TYPE;**

**v\_email UZYTKOWNIK.EMAIL%TYPE;**

**v\_telefon UZYTKOWNIK.TELEFON%TYPE;**

**v\_stan\_konta UZYTKOWNIK.STAN\_KONTA%TYPE;**

**BEGIN**

**-- 1. Pobieramy dane starego rekordu do zmiennych**

**SELECT IMIE, NAZWISKO, PESEL, DATA\_REJESTRACJI, EMAIL, TELEFON, STAN\_KONTA**

**INTO v\_imie, v\_nazwisko, v\_pesel, v\_data\_rej, v\_email, v\_telefon, v\_stan\_konta**

**FROM UZYTKOWNIK**

**WHERE UZYTKOWNIK\_ID = p\_uzytkownik\_id;**

**-- 2. Wrzucamy je do tabeli ARCHIWUM w formacie JSON**

**INSERT INTO ARCHIWUM (**

**NAZWA\_TABLICY,**

**ID\_W\_PIERWOTNEJ,**

**DANE\_JSON,**

**DATA\_ARCHIWIZACJI**

**)**

**VALUES (**

**'UZYTKOWNIK',**

**p\_uzytkownik\_id,**

**'{"IMIE":"' || v\_imie ||**

**'","NAZWISKO":"' || v\_nazwisko ||**

**'","PESEL":"' || v\_pesel ||**

**'","DATA\_REJESTRACJI":"' || TO\_CHAR(v\_data\_rej, 'YYYY-MM-DD') ||**

**'","EMAIL":"' || v\_email ||**

**'","TELEFON":"' || v\_telefon ||**

**'","STAN\_KONTA":"' || v\_stan\_konta || '"}',**

**SYSDATE**

**);**

**-- 3. Usuwamy rekord z tabeli UZYTKOWNIK**

**DELETE FROM UZYTKOWNIK**

**WHERE UZYTKOWNIK\_ID = p\_uzytkownik\_id;**

**END;**

**/**

Tworzenie zakładu

**create or replace PROCEDURE STWORZ\_LUB\_EDYTUJ\_ZAKLAD(**

**p\_zaklad\_id IN NUMBER, -- NULL => nowy zakład**

**p\_nazwa IN VARCHAR2,**

**p\_dyscyplina IN VARCHAR2,**

**p\_data\_zdarzenia IN DATE,**

**p\_kurs IN NUMBER**

**)**

**IS**

**BEGIN**

**IF p\_zaklad\_id IS NULL THEN**

**-- INSERT nowego zakładu**

**INSERT INTO ZAKLAD (**

**NAZWA,**

**DYSCYPLINA,**

**DATA\_ZDARZENIA,**

**KURS**

**)**

**VALUES (**

**p\_nazwa,**

**p\_dyscyplina,**

**p\_data\_zdarzenia,**

**p\_kurs**

**);**

**ELSE**

**-- UPDATE istniejącego zakładu**

**UPDATE ZAKLAD**

**SET NAZWA = p\_nazwa,**

**DYSCYPLINA = p\_dyscyplina,**

**DATA\_ZDARZENIA = p\_data\_zdarzenia,**

**KURS = p\_kurs**

**WHERE ZAKLAD\_ID = p\_zaklad\_id;**

**END IF;**

**END;**

Usuwanie zakładu:

**create or replace PROCEDURE USUN\_ZAKLAD(**

**p\_zaklad\_id IN NUMBER**

**)**

**IS**

**v\_nazwa ZAKLAD.NAZWA%TYPE;**

**v\_dyscyplina ZAKLAD.DYSCYPLINA%TYPE;**

**v\_data\_zdarzenia ZAKLAD.DATA\_ZDARZENIA%TYPE;**

**v\_kurs ZAKLAD.KURS%TYPE;**

**BEGIN**

**-- 1. Pobieramy dane z ZAKLAD**

**SELECT NAZWA, DYSCYPLINA, DATA\_ZDARZENIA, KURS**

**INTO v\_nazwa, v\_dyscyplina, v\_data\_zdarzenia, v\_kurs**

**FROM ZAKLAD**

**WHERE ZAKLAD\_ID = p\_zaklad\_id;**

**-- 2. Zapisujemy w ARCHIWUM**

**INSERT INTO ARCHIWUM (**

**NAZWA\_TABLICY,**

**ID\_W\_PIERWOTNEJ,**

**DANE\_JSON,**

**DATA\_ARCHIWIZACJI**

**)**

**VALUES (**

**'ZAKLAD',**

**p\_zaklad\_id,**

**'{"NAZWA":"' || v\_nazwa ||**

**'","DYSCYPLINA":"' || v\_dyscyplina ||**

**'","DATA\_ZDARZENIA":"' || TO\_CHAR(v\_data\_zdarzenia, 'YYYY-MM-DD') ||**

**'","KURS":"' || v\_kurs || '"}',**

**SYSDATE**

**);**

**-- 3. Usuwamy rekord z ZAKLAD**

**DELETE FROM ZAKLAD**

**WHERE ZAKLAD\_ID = p\_zaklad\_id;**

**END;**

Tworzenie procedury:

**CREATE OR REPLACE PROCEDURE STWORZ\_LUB\_EDYTUJ\_TRANSAKCJE(**

**p\_transakcja\_id IN NUMBER, -- NULL => nowa transakcja**

**p\_uzytkownik\_id IN NUMBER,**

**p\_rodzaj IN VARCHAR2,**

**p\_kwota IN NUMBER,**

**p\_data\_transakcji IN DATE,**

**p\_zaklad\_id IN NUMBER -- może być NULL (np. przy WPŁACIE)**

**)**

**IS**

**BEGIN**

**IF p\_transakcja\_id IS NULL THEN**

**-- INSERT nowej transakcji**

**INSERT INTO TRANSAKCJA (**

**UZYTKOWNIK\_ID,**

**RODZAJ,**

**KWOTA,**

**DATA\_TRANSAKCJI,**

**ZAKLAD\_ID**

**)**

**VALUES (**

**p\_uzytkownik\_id,**

**p\_rodzaj,**

**p\_kwota,**

**p\_data\_transakcji,**

**p\_zaklad\_id**

**);**

**ELSE**

**-- UPDATE istniejącej transakcji**

**UPDATE TRANSAKCJA**

**SET UZYTKOWNIK\_ID = p\_uzytkownik\_id,**

**RODZAJ = p\_rodzaj,**

**KWOTA = p\_kwota,**

**DATA\_TRANSAKCJI = p\_data\_transakcji,**

**ZAKLAD\_ID = p\_zaklad\_id**

**WHERE TRANSAKCJA\_ID = p\_transakcja\_id;**

**END IF;**

**END;**

**/**

Usuwanie transakcji:

**CREATE OR REPLACE PROCEDURE USUN\_TRANSAKCJE(**

**p\_transakcja\_id IN NUMBER**

**)**

**IS**

**v\_uzytkownik\_id TRANSAKCJA.UZYTKOWNIK\_ID%TYPE;**

**v\_rodzaj TRANSAKCJA.RODZAJ%TYPE;**

**v\_kwota TRANSAKCJA.KWOTA%TYPE;**

**v\_data\_transakcji TRANSAKCJA.DATA\_TRANSAKCJI%TYPE;**

**v\_zaklad\_id TRANSAKCJA.ZAKLAD\_ID%TYPE;**

**BEGIN**

**-- 1. Pobieramy dane z TRANSAKCJA**

**SELECT UZYTKOWNIK\_ID,**

**RODZAJ,**

**KWOTA,**

**DATA\_TRANSAKCJI,**

**ZAKLAD\_ID**

**INTO v\_uzytkownik\_id,**

**v\_rodzaj,**

**v\_kwota,**

**v\_data\_transakcji,**

**v\_zaklad\_id**

**FROM TRANSAKCJA**

**WHERE TRANSAKCJA\_ID = p\_transakcja\_id;**

**-- 2. Archiwizujemy**

**INSERT INTO ARCHIWUM (**

**NAZWA\_TABLICY,**

**ID\_W\_PIERWOTNEJ,**

**DANE\_JSON,**

**DATA\_ARCHIWIZACJI**

**)**

**VALUES (**

**'TRANSAKCJA',**

**p\_transakcja\_id,**

**'{"UZYTKOWNIK\_ID":"' || v\_uzytkownik\_id ||**

**'","RODZAJ":"' || v\_rodzaj ||**

**'","KWOTA":"' || v\_kwota ||**

**'","DATA\_TRANSAKCJI":"' || TO\_CHAR(v\_data\_transakcji, 'YYYY-MM-DD') ||**

**'","ZAKLAD\_ID":"' || v\_zaklad\_id || '"}',**

**SYSDATE**

**);**

**-- 3. Usuwamy rekord z TRANSAKCJA**

**DELETE FROM TRANSAKCJA**

**WHERE TRANSAKCJA\_ID = p\_transakcja\_id;**

**END;**

**/**

Wpłacanie (i potencjalne wypłacanie) środków na konto:

**CREATE OR REPLACE PROCEDURE AKTUALIZUJ\_STAN\_KONTA (**

**p\_uzytkownik\_id IN NUMBER, -- ID użytkownika, którego konto aktualizujemy**

**p\_kwota IN NUMBER -- wartość zmiany stanu konta (może być dodatnia lub ujemna)**

**)**

**IS**

**v\_stan\_konta NUMBER(10,2);**

**BEGIN**

**-- 1. Sprawdzamy, czy dany użytkownik istnieje i pobieramy jego obecne saldo**

**SELECT STAN\_KONTA**

**INTO v\_stan\_konta**

**FROM UZYTKOWNIK**

**WHERE UZYTKOWNIK\_ID = p\_uzytkownik\_id;**

**-- 2. (Opcjonalnie) Sprawdzamy, czy saldo nie spadnie poniżej zera, jeśli np. to nie jest dozwolone**

**IF (v\_stan\_konta + p\_kwota) < 0 THEN**

**RAISE\_APPLICATION\_ERROR(-20002, 'Operacja niedozwolona: saldo nie może być ujemne.');**

**END IF;**

**-- 3. Aktualizujemy stan konta**

**UPDATE UZYTKOWNIK**

**SET STAN\_KONTA = STAN\_KONTA + p\_kwota**

**WHERE UZYTKOWNIK\_ID = p\_uzytkownik\_id;**

**IF SQL%ROWCOUNT = 0 THEN**

**-- Jeśli nie zmieniliśmy żadnego wiersza, tzn. użytkownik nie istnieje**

**RAISE\_APPLICATION\_ERROR(-20001, 'Nie znaleziono użytkownika o podanym ID.');**

**END IF;**

**-- 4. (Opcjonalnie) Logujemy operację w tabeli LOGI lub innej**

**INSERT INTO LOGI (DATA\_OPERACJI, OPIS)**

**VALUES (SYSDATE,**

**'Zaktualizowano stan konta użytkownika ID=' || p\_uzytkownik\_id ||**

**' o kwotę=' || p\_kwota);**

**COMMIT; -- potwierdzenie zmian w bazie**

**EXCEPTION**

**WHEN NO\_DATA\_FOUND THEN**

**-- Obsługa sytuacji, gdy SELECT nie zwróci żadnego wiersza**

**RAISE\_APPLICATION\_ERROR(-20003, 'Użytkownik o podanym ID nie istnieje.');**

**WHEN OTHERS THEN**

**-- Obsługa nieoczekiwanych błędów**

**ROLLBACK;**

**RAISE;**

**END;**

**/**

Procedura wypłacania pieniędzy:

**CREATE OR REPLACE PROCEDURE WYPLAC\_ZAKLAD (  
 p\_transakcja\_id IN NUMBER  
 )  
 IS  
 v\_uzytkownik\_id NUMBER;  
 v\_zaklad\_id NUMBER;  
 v\_kwota\_stawki NUMBER(10,2);  
 v\_kurs NUMBER(5,2);  
 v\_status\_zakladu VARCHAR2(20);  
 v\_kwota\_wygranej NUMBER(10,2);  
 BEGIN  
 ------------------------------------------------------------------------------  
 -- 1. Pobieramy dane o obstawieniu zakładu (stawka, zaklad\_id, użytkownik\_id)  
 ------------------------------------------------------------------------------  
 SELECT UZYTKOWNIK\_ID,  
 ZAKLAD\_ID,  
 KWOTA  
 INTO v\_uzytkownik\_id,  
 v\_zaklad\_id,  
 v\_kwota\_stawki  
 FROM TRANSAKCJA  
 WHERE TRANSAKCJA\_ID = p\_transakcja\_id  
 AND RODZAJ = 'ZAKŁAD';**

**------------------------------------------------------------------------------  
 -- 2. Pobieramy kurs i status zakładu z tabeli ZAKLAD  
 ------------------------------------------------------------------------------  
 SELECT KURS, STATUS  
 INTO v\_kurs, v\_status\_zakladu  
 FROM ZAKLAD  
 WHERE ZAKLAD\_ID = v\_zaklad\_id;**

**------------------------------------------------------------------------------  
 -- 3. Sprawdzamy, czy zakład ma status "WYGRANY"  
 ------------------------------------------------------------------------------  
 IF v\_status\_zakladu <> 'WYGRANY' THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Zakład nie jest wygrany. Wypłata niemożliwa.');  
 END IF;**

**------------------------------------------------------------------------------  
 -- 4. Obliczamy kwotę wygranej  
 ------------------------------------------------------------------------------  
 v\_kwota\_wygranej := v\_kwota\_stawki \* v\_kurs;**

**------------------------------------------------------------------------------  
 -- 5. Zwiększamy stan konta użytkownika w tabeli UZYTKOWNIK  
 ------------------------------------------------------------------------------  
 UPDATE UZYTKOWNIK  
 SET STAN\_KONTA = STAN\_KONTA + v\_kwota\_wygranej  
 WHERE UZYTKOWNIK\_ID = v\_uzytkownik\_id;**

**IF SQL%ROWCOUNT = 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Nie znaleziono użytkownika do wypłaty wygranej.');  
 END IF;**

**------------------------------------------------------------------------------  
 -- 6. Rejestrujemy w tabeli TRANSAKCJA nowy wpis typu "WYGRANA"  
 ------------------------------------------------------------------------------  
 INSERT INTO TRANSAKCJA (  
 UZYTKOWNIK\_ID,  
 RODZAJ,  
 KWOTA,  
 DATA\_TRANSAKCJI,  
 ZAKLAD\_ID  
 )  
 VALUES (  
 v\_uzytkownik\_id,  
 'WYGRANA',  
 v\_kwota\_wygranej,  
 SYSDATE,  
 v\_zaklad\_id  
 );**

**------------------------------------------------------------------------------  
 -- 7. (Opcjonalnie) logujemy operację w tabeli LOGI i potwierdzamy transakcję  
 ------------------------------------------------------------------------------  
 INSERT INTO LOGI (DATA\_OPERACJI, OPIS)  
 VALUES (SYSDATE,  
 'Wypłata wygranej z transakcji='||p\_transakcja\_id||  
 ' (zakład='||v\_zaklad\_id||  
 '), kwota='||v\_kwota\_wygranej);**

**COMMIT;**

**EXCEPTION  
 WHEN NO\_DATA\_FOUND THEN  
 -- Jeśli SELECT nie znajdzie transakcji typu ZAKŁAD lub zakładu w tabeli ZAKLAD  
 ROLLBACK;  
 RAISE\_APPLICATION\_ERROR(-20003, 'Nie znaleziono transakcji typu "ZAKŁAD" lub zakładu.');  
 WHEN OTHERS THEN  
 ROLLBACK;  
 RAISE;  
 END;  
 /**

### Archiwizacja usuniętych danych

Zgodnie z założeniami, usunięte dane (np. z tabel UZYTKOWNIK, TRANSAKCJA) trafiają do tabeli ARCHIWUM w formacie JSON.

**Wyzwalacz** BEFORE DELETE na tabeli UZYTKOWNIK:

**CREATE OR REPLACE TRIGGER TRG\_UZYTKOWNIK\_ARCHIWUM**

**BEFORE DELETE**

**ON UZYTKOWNIK**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO ARCHIWUM (NAZWA\_TABLICY, ID\_W\_PIERWOTNEJ, DANE\_JSON)**

**VALUES (**

**'UZYTKOWNIK',**

**:OLD.UZYTKOWNIK\_ID,**

**'{"IMIE":"'||:OLD.IMIE||'",**

**"NAZWISKO":"'||:OLD.NAZWISKO||'",**

**"PESEL":"'||:OLD.PESEL||'",**

**"DATA\_REJESTRACJI":"'||TO\_CHAR(:OLD.DATA\_REJESTRACJI, 'YYYY-MM-DD')||'",**

**"EMAIL":"'||:OLD.EMAIL||'",**

**"TELEFON":"'||:OLD.TELEFON||'",**

**"STAN\_KONTA":"'||:OLD.STAN\_KONTA||'"}'**

**);**

**END;**

**/**

Dzięki temu, nawet jeśli rekord zostanie fizycznie usunięty z tabeli UZYTKOWNIK, w tabeli ARCHIWUM zostanie zapisana jego historia.

### Logowanie informacji do tabeli

Możemy stworzyć prostą tabelę LOGI do przechowywania komunikatów o wykonanych działaniach, np.:

**CREATE TABLE LOGI (**

**LOG\_ID NUMBER GENERATED BY DEFAULT AS IDENTITY,**

**DATA\_OPERACJI DATE DEFAULT SYSDATE,**

**OPIS VARCHAR2(4000)**

**);**

Następnie np. w **procedurach** aktualizujących stan konta użytkownika możemy dodać wpis do LOGI:

**CREATE OR REPLACE PROCEDURE ZAKTUALIZUJ\_STAN\_KONTA(**

**p\_uzytkownik\_id IN NUMBER,**

**p\_kwota IN NUMBER**

**)**

**AS**

**BEGIN**

**UPDATE UZYTKOWNIK**

**SET STAN\_KONTA = STAN\_KONTA + p\_kwota**

**WHERE UZYTKOWNIK\_ID = p\_uzytkownik\_id;**

**INSERT INTO LOGI(OPIS)**

**VALUES('Zaktualizowano stan konta użytkownika ID=' || p\_uzytkownik\_id ||**

**' o kwotę=' || p\_kwota);**

**END;**

**/**

### Obsługa wyjątków, również własne wyjątki

W PL/SQL możemy definiować **własne wyjątki**, które sygnalizują nietypowe sytuacje biznesowe. Np. gdy użytkownik próbuje obstawić zakład, ale nie ma wystarczających środków na koncie:

**CREATE OR REPLACE PROCEDURE OBSTAW\_ZAKLAD(**

**p\_uzytkownik\_id IN NUMBER,**

**p\_zaklad\_id IN NUMBER,**

**p\_kwota IN NUMBER**

**)**

**IS**

**ex\_brak\_srodkow EXCEPTION;**

**v\_stan\_konta NUMBER(10,2);**

**BEGIN**

**-- pobieramy stan konta**

**SELECT STAN\_KONTA INTO v\_stan\_konta**

**FROM UZYTKOWNIK**

**WHERE UZYTKOWNIK\_ID = p\_uzytkownik\_id;**

**-- sprawdzamy czy jest wystarczająca ilość środków**

**IF v\_stan\_konta < p\_kwota THEN**

**RAISE ex\_brak\_srodkow;**

**END IF;**

**-- jeśli środków jest dość, to kontynuujemy**

**INSERT INTO TRANSAKCJA(UZYTKOWNIK\_ID, RODZAJ, KWOTA, ZAKLAD\_ID)**

**VALUES(p\_uzytkownik\_id, 'ZAKŁAD', p\_kwota, p\_zaklad\_id);**

**-- pomniejszamy stan konta**

**UPDATE UZYTKOWNIK**

**SET STAN\_KONTA = STAN\_KONTA - p\_kwota**

**WHERE UZYTKOWNIK\_ID = p\_uzytkownik\_id;**

**INSERT INTO LOGI(OPIS)**

**VALUES('Użytkownik ID=' || p\_uzytkownik\_id || ' obstawił zakład ID='**

**|| p\_zaklad\_id || ' kwotą=' || p\_kwota);**

**EXCEPTION**

**WHEN ex\_brak\_srodkow THEN**

**INSERT INTO LOGI(OPIS)**

**VALUES('Błąd: użytkownik ID=' || p\_uzytkownik\_id**

**|| ' nie ma wystarczających środków.');**

**RAISE\_APPLICATION\_ERROR(-20002, 'Brak środków na koncie!');**

**END;**

**/**

### Procedury, funkcje z parametrami, możliwe parametry domyślne, wykorzystanie funkcji okienkowych

Możemy stworzyć procedurę, która ma parametry domyślne, np. generowanie raportu w zależności od **okresu**:

**CREATE OR REPLACE PROCEDURE GENERUJ\_RAPORT(**

**p\_okres IN VARCHAR2 DEFAULT 'MIESIAC' -- MIESIAC, KWARTAL lub ROK**

**)**

**AS**

**BEGIN**

**IF p\_okres = 'MIESIAC' THEN**

**-- generuj raport miesięczny**

**INSERT INTO LOGI(OPIS) VALUES('Generowanie raportu miesięcznego...');**

**ELSIF p\_okres = 'KWARTAL' THEN**

**-- generuj raport kwartalny**

**INSERT INTO LOGI(OPIS) VALUES('Generowanie raportu kwartalnego...');**

**ELSIF p\_okres = 'ROK' THEN**

**-- generuj raport roczny**

**INSERT INTO LOGI(OPIS) VALUES('Generowanie raportu rocznego...');**

**ELSE**

**RAISE\_APPLICATION\_ERROR(-20003, 'Nieznany zakres raportu: ' || p\_okres);**

**END IF;**

**END;**

/

## **Podsumowanie**

1. **Tworzenie/Edycja**
   * W każdej z głównych tabel (UZYTKOWNIK, ZAKLAD, TRANSAKCJA) mamy procedurę, która w zależności od tego, czy podano identyfikator rekordu (ID), wykonuje INSERT (dla nowego) lub UPDATE (dla istniejącego).
2. **Miękkie usuwanie bez kolumny IS\_DELETED**
   * Zamiast ustawiać flagę (IS\_DELETED), **fizycznie usuwamy** rekord z tabeli i **zapisujemy** jego zawartość w tabeli ARCHIWUM (np. w formacie JSON).
   * Dzięki temu mamy historyczne dane, które można w razie potrzeby przywrócić lub przeglądać.
3. **Archiwizacja**
   * Tabela ARCHIWUM ma kolumnę z nazwą tabeli źródłowej (NAZWA\_TABLICY), identyfikatorem usuniętego rekordu (ID\_W\_PIERWOTNEJ), pole DANE\_JSON do przechowywania szczegółów oraz znacznik czasu archiwizacji (DATA\_ARCHIWIZACJI).
4. **Rozszerzenia**
   * Można wzbogacić procedury o **obsługę wyjątków** (sekcję EXCEPTION), logowanie do tabeli LOGI, a także dodatkową weryfikację danych (np. czy stan konta jest wystarczający, czy kurs jest w dozwolonym zakresie, itp.).

### Sprawdzanie poprawności dodawanych danych (np. funkcja sprawdzająca poprawność pesela)

**Walidacja PESEL**

Chcemy mieć pewność, że PESEL ma poprawną długość (11 znaków) i – opcjonalnie – zgodny z regułami obliczania sumy kontrolnej.

Przykładowa funkcja w PL/SQL, która weryfikuje PESEL tylko pod kątem długości (dla przykładu, bo pełna walidacja jest nieco dłuższa):

**CREATE OR REPLACE FUNCTION SPRAWDZ\_PESEL (**

**p\_pesel IN VARCHAR2**

**) RETURN BOOLEAN**

**IS**

**BEGIN**

**IF LENGTH(p\_pesel) = 11 THEN**

**RETURN TRUE;**

**ELSE**

**RETURN FALSE;**

**END IF;**

**END;**

**/**

Możemy tę funkcję wykorzystywać w wyzwalaczu BEFORE INSERT na tabeli UZYTKOWNIK:

**CREATE OR REPLACE TRIGGER TRG\_UZYTKOWNIK\_CHECK\_PESEL**

**BEFORE INSERT OR UPDATE**

**ON UZYTKOWNIK**

**FOR EACH ROW**

**BEGIN**

**IF NOT SPRAWDZ\_PESEL(:NEW.PESEL) THEN**

**RAISE\_APPLICATION\_ERROR(-20001, 'Niepoprawny PESEL!');**

**END IF;**

**END;**

**/**

# Procedury, funkcje, wyzwalacze tworzące podsumowania

### **Parametry domyślne**

CREATE OR REPLACE PROCEDURE GENERUJ\_RAPORT(

p\_okres IN VARCHAR2 DEFAULT 'MIESIAC' -- MIESIAC, KWARTAL lub ROK

)

AS

BEGIN

IF p\_okres = 'MIESIAC' THEN

-- generuj raport miesięczny

INSERT INTO LOGI(OPIS) VALUES('Generowanie raportu miesięcznego...');

ELSIF p\_okres = 'KWARTAL' THEN

-- generuj raport kwartalny

INSERT INTO LOGI(OPIS) VALUES('Generowanie raportu kwartalnego...');

ELSIF p\_okres = 'ROK' THEN

-- generuj raport roczny

INSERT INTO LOGI(OPIS) VALUES('Generowanie raportu rocznego...');

ELSE

RAISE\_APPLICATION\_ERROR(-20003, 'Nieznany zakres raportu: ' || p\_okres);

END IF;

END;

/

Możemy stworzyć procedurę, która ma parametry domyślne, np. generowanie raportu w zależności od **okresu**:

Nasi przełożeni mogą zażądać **zestawień** w różnych ujęciach (miesięcznych, kwartalnych, rocznych). Wygenerowane wyniki zapisujemy w **tabeli** (np. RAPORTY), aby były gotowe do wizualizacji (np. na wykresach w aplikacji front-end).

Przykładowa tabela:

CREATE TABLE RAPORTY (

RAPORT\_ID NUMBER GENERATED BY DEFAULT AS IDENTITY,

OKRES VARCHAR2(50),

SUMA\_ZAKLADOW NUMBER(15,2),

DATA\_RAPORTU DATE DEFAULT SYSDATE,

PRIMARY KEY(RAPORT\_ID)

);

CREATE OR REPLACE PROCEDURE GENERUJ\_RAPORT\_OKRESOWY(

p\_okres IN VARCHAR2 -- 'MIESIAC', 'KWARTAL', 'ROK'

)

AS

v\_suma NUMBER(15,2);

BEGIN

IF p\_okres = 'MIESIAC' THEN

SELECT SUM(KWOTA) INTO v\_suma

FROM TRANSAKCJA

WHERE RODZAJ = 'ZAKŁAD'

AND DATA\_TRANSAKCJI >= ADD\_MONTHS(TRUNC(SYSDATE, 'MONTH'), 0)

AND DATA\_TRANSAKCJI < ADD\_MONTHS(TRUNC(SYSDATE, 'MONTH'), 1);

ELSIF p\_okres = 'KWARTAL' THEN

SELECT SUM(KWOTA) INTO v\_suma

FROM TRANSAKCJA

WHERE RODZAJ = 'ZAKŁAD'

AND DATA\_TRANSAKCJI >= ADD\_MONTHS(TRUNC(SYSDATE, 'Q'), 0)

AND DATA\_TRANSAKCJI < ADD\_MONTHS(TRUNC(SYSDATE, 'Q'), 3);

ELSIF p\_okres = 'ROK' THEN

SELECT SUM(KWOTA) INTO v\_suma

FROM TRANSAKCJA

WHERE RODZAJ = 'ZAKŁAD'

AND DATA\_TRANSAKCJI >= TRUNC(SYSDATE, 'YEAR')

AND DATA\_TRANSAKCJI < ADD\_MONTHS(TRUNC(SYSDATE, 'YEAR'), 12);

ELSE

RAISE\_APPLICATION\_ERROR(-20004, 'Nieznany okres raportu: ' || p\_okres);

END IF;

INSERT INTO RAPORTY(OKRES, SUMA\_ZAKLADOW)

VALUES(p\_okres, NVL(v\_suma, 0));

END;

/

Procedura powyżej wylicza łączną kwotę postawionych zakładów w zależności od wybranego okresu (miesiąc, kwartał, rok), a następnie zapisuje ten wynik w tabeli RAPORTY.