**PERSPEKTYWY**

**//zad 1** Utwórz perspektywę o nazwie STATYSTYKA\_klentów, która będzie przedstawiać imię, nazwisko klienta oraz liczbę dokonanych przez niego wypożyczeni.

create view STATYSTYKA\_KLIENTOW

as select imie, nazwisko, count (\*) as l\_wypozyczen

from klienci

natural join wypozyczenia

group by imie, nazwisko

select \* from STATYSTYKA\_KLIENTOW

**//zad 2** Korzystając z perspektywy STATYSTYKA\_KLIENTOW wyświetl imiona i nazwiska klientów, którzy wykonali ponad 2 wypożyczenia.

select \* from STATYSTYKA\_KLIENTOW

where l\_wypozyczen>2

**//zad3** Usuń perspektyę STATYSTYKA\_KLIENTOW, wysświetl imiona i nazwiska kientów, którzy wykonali ponad 2 wypożyczenia.

drop view STATYSTYKA\_KLIENTOW;

Usunięcie perspektywy nie powoduje utraty danych, ponieważ

perspektywa jest relacją wirtualną i nie zawiera rzeczywistych

krotek.

**TRANSAKCJE**

**//zad1** Rozpocznij nową transakcję, usuń wszystkie krotki z relacji FILMY, a następnie wycofaj bieżącą transakcje. Sprawdź zawartość relacji filmy.

begin;

delete from filmy;

rollback;

select \* from filmy;

**//zad2** Wykonaj następujące kroki.

1. Rozpocznij nową transakcję.
2. Do relacji filmy wprowadź nową krotkę ID\_FILMU=11, TYTUL=’Komornik’, ROK\_PRODUKCJI=2005, CENA=10.5.
3. Utwórz punkt zachowania o nazwie P1
4. Usuń wszystkie krotki z relacji FILMY
5. Utwórz punkt zachowania o nazwie P2
6. Usuń wszystkie krotki z relacji KOPIE
7. Wyświetl zawartość relacji FILMY i KOPIE
8. Wycofaj transakcje do punktu zachowania P2
9. Wyświetl zawartość relacji FILMY i KOPIE
10. Wycofaj transakcje do punktu zachowania P1
11. Wyświetl zawartość relacji FILMY i KOPIE
12. Wycofaj transakcję
13. Wyświetl zawartość relacji FILMY i KOPIE

begin;

insert into filmy (id\_filmu,tytul,rok\_produkcji,cena) values (11, 'Komornik', 2005,10.5);

savepoint p1;

delete from filmy;

savepoint p2;

delete from kopie;

select \* from filmy;

select \* from kopie;

rollback to p2;

select \* from filmy;

select \* from kopie;

rollback to p1;

select \* from filmy;

select \* from kopie;

rollback;

select \* from filmy;

select \* from kopie;

**//zad3** Podłącz się do bazy danych równocześnie z dwóch okien aplikacji. Wykonaj następujące kroki.

1. W pierwszym oknie rozpocznij nową transakcję.
2. W pierwszym oknie do relacji FILMY wprowadź nową krotkę ID\_FILMY=11, TYTUL=’Komornik’, ROK\_PRODUKCJI=2005, CENA=10.5
3. W pierwszym oknie wyświetl zawartość relacji FILMY
4. W drugim oknie wyświetl zawartość relacji filmy.
5. W pierwszym oknie zatwierdź transakcje
6. W pierwszym oknie wyświetl zawartość relacji FILMY.
7. W drugim oknie wyświetl zawartość relacji FILMY.

begin;

insert into filmy (id\_filmu,tytul,rok\_produkcji,cena) values (11, 'Komornik', 2005,10.5);

select \* from filmy; --- w pierwszysm oknie wyświetla się id 11, w drugim nie---

commit;

select \* from filmy;---w obydwuch oknach id 11 wyświetla się----

**Drugie okno aplikacji**

select \* from filmy;

rollback;

select \* from filmy;

**//zad4 Podłącz się do bazy danych równocześnie z dwóch okien aplikacji. Wykonaj następujące kroki:**

1. **W pierwszym oknie rozpocznij nową transakcje.**
2. **W drugim oknie rozpocznij nową transakcje**
3. **W pierwszym oknie zmień cenę filmu ‘Wesele’ na 15zł**
4. **W drugim oknie zmień rok produkcji ‘Ronin’ na 2000**
5. **W pierwszym oknie zmień cenę ‘Ronin’ na 15 zł**
6. **W drugim oknie zmień rok produkcji ‘Wesele’ na 2000. Co się stało? Dlaczego?**

rollback;

begin;

update filmy set cena=15 where tytul='Wesele';

update filmy set cena=15 where tytul='Ronin';

select \* from filmy;

---error deadlock, blokada krotek---

**Drugie okno aplikacji**

rollback;

begin;

update filmy set rok\_produkcji=2000 where tytul='Wesele';

update filmy set rok\_produkcji=2000 where tytul='Ronin';

select \* from filmy;

---error deadlock, blokada krotek---

1. **Połącz się z bazą danych jako administrator (postgres). Utwórz dwa nowe konta użytkowników: a. nazwa użytkownika: jacek, hasło:123 b. nazwa użytkownika: placek, hasło: 456 Jako odpowiedzi rejestruj polecenia wymagane przez poszczególne zadania**

create user jacek password '123';

create user placek password '456';

1. **Połącz się z bazą danych jako użytkownik jacek. Utwórz nową relację TELEFONY o następującym schemacie: TELEFONY(ID\_OSOBY, IMIE, NUMER\_TEL). Wprowadź trzy dowolne krotki do relacji TELEFONY.**

create table TELEFONY

(ID\_OSOBY serial primary key,

IMIE varchar(50) not null,

NUMER\_TEL varchar(9));

1. **Połącz się z bazą danych jako użytkownik placek. Spróbuj wyświetlić wszystkie krotki relacji TELEFONY. Co się stało? Dlaczego? Jako odpowiedź, oprócz poleceń załącz także odpowiedzi na postawione pytania.**

select \* from telefony

Nie da się, bo placek nie ma dostępu do tej tabeli

1. **Połącz się z bazą danych jako użytkownik jacek. Przekaż użytkownikowi placek prawa do odczytu i wprowadzania krotek do relacji TELEFONY.**

GRANT INSERT ON telefony TO placek;

GRANT SELECT ON telefony TO placek;

1. **Połącz się z bazą danych jako użytkownik placek. Spróbuj wyświetlić wszystkie krotki relacji TELEFONY. Spróbuj wprowadzić nową krotkę do relacji TELEFONY. Spróbuj zmienić numer telefonu w dowolnej krotce relacji TELEFONY. Co się stało? Dlaczego?**

select \* from telefony

insert into telefony (id\_osoby, imie, numer\_tel) values(4, 'Anna', '723123123')

set imie = 'Jan' where id\_osoby = 1

--- ERROR: permission denied for table telefony

1. **Połącz się z bazą danych jako użytkownik jacek. Odbierz użytkownikowi placek prawo do wprowadzania krotek do relacji TELEFONY.**

REVOKE INSERT ON telefony FROM placek;

**7. Połącz się z bazą danych jako użytkownik placek. Spróbuj wyświetlić wszystkie krotki relacji TELEFONY. Spróbuj wprowadzić nową krotkę do relacji TELEFONY. Co się stało? Dlaczego?**

insert into telefony (id\_osoby, imie, numer\_tel) values (5, 'Anna', '723123123')

--- ERROR: permission denied for table telefony

**4.INDEKSY**

**1. Utwórz relację PIASEK (ID\_ZIARENKA SERIAL, MASA\_ZIARENKA REAL, KOLOR\_ZIARENKA VARCHAR(10)). Wprowadź cztery dowolne krotki do relacji PIASEK. Np.:**

Obraz zawierający tekst

Opis wygenerowany automatycznie

create table PIASEK

(ID\_ZIARENKA SERIAL, MASA\_ZIARENKA REAL,

KOLOR\_ZIARENKA VARCHAR(10));

insert into PIASEK (MASA\_ZIARENKA,

KOLOR\_ZIARENKA) values

(0.025, 'szary'),

(0.021, 'czarny'),

(0.018, 'bialy'),

(0.031, 'czarny');

**2. Korzystając z polecenia INSERT INTO SELECT powielaj krotki relacji PIASEK tyle razy, aby posiadała ponad milion krotek. Uwaga: nie zamazuj wartości automatycznie wypełnianego atrybutu ID\_ZIARENKA. select count(\*) from piasek; 1048576**

insert into PIASEK (masa\_ziarenka,kolor\_ziarenka)

select masa\_ziarenka,kolor\_ziarenka from PIASEK;

select count (\*) from PIASEK;

**3. Dokonaj pomiaru czasu wykonania zapytania wyświetlającego kolor ziarenka piasku o identyfikatorze 123456. W odpowiedzi zanotuj uzyskany czas.**

select kolor\_ziarenka from PIASEK where id\_ziarenka=123456;

**4.Dokonaj pomiaru czasu wykonania zapytania wyświetlającego liczbę ziarenek piasku o identyfikatorach pomiędzy 500000 a 500005.**

select count(\*) from PIASEK where id\_ziarenka>=500000 and id\_ziarenka<=500005;

**5.Dokonaj pomiaru czasu wykonania zapytania wyświetlającego liczbę ziarenek piasku o masie 0.025 g**.

select count(\*) from PIASEK where masa\_ziarenka=0.025

**6.Utwórz indeks B\*-drzewo o nazwie PIASEK\_IND oparty na atrybucie ID\_ZIARENKA relacji PIASEK.**

create index PIASEK\_INX on PIASEK(ID\_ziarenka);

**7. Ponownie dokonaj pomiaru czasu wykonania zapytania wyświetlającego kolor ziarenka piasku o identyfikatorze 123456. Czy czas ten uległ zmianie? Dlaczego?**

select kolor\_ziarenka from PIASEK where id\_ziarenka=123456; z indeksami jest szybciej

**8. Dokonaj pomiaru czasu wykonania zapytania wyświetlającego liczbę ziarenek piasku o identyfikatorach pomiędzy 500000 a 500005. Czy czas ten uległ zmianie? Dlaczego?**

select count(\*) from piasek where id\_ziarenka BETWEEN 500000 and 500005

**9. Dokonaj pomiaru czasu wykonania zapytania wyświetlającego liczbę ziarenek piasku o masie 0.025 g. Czy czas ten uległ zmianie? Dlaczego?**

select count(\*) from piasek where masa\_ziarenka = '0.025'

10. Usuń indeks B\*-drzewo o nazwie PIASEK\_IND.

DROP INDEX piasek\_ind

ZJAZD 5 – 5 temat PL/PGSQL

**1. Utwórz funkcję o nazwie CENA\_BRUTTO(c REAL), która dla podanej ceny netto wyliczy odpowiadającą jej cenę brutto, powiększoną o 23%. Następnie wykonaj zapytanie, które na podstawie relacji FILMY wyświetli: tytuł filmu, cenę (netto) i cenę brutto.**

create function cena\_brutto ( c real)

returns real as $$

declare wynik real;

begin wynik:=c\*1.23;

return wynik;

end;

$$ language PLPGSQL;

--test

select tytul, cena as cena\_netto, cena\_brutto(cena) from filmy

**2. Utwórz funkcję o nazwie OPIS\_FILMU(id INTEGER), która dla podanego identyfikatora filmu wyświetli jego opis zgodnie z poniższym wzorcem: SELECT OPIS\_FILMU(1); Film “Ghostbusters” zostal nakrecony w roku 1984 przy udziale aktorow: Bill Murray, Dan Aykroyd, Sigourney Weaver.**

CREATE OR REPLACE FUNCTION opis\_filmu(id integer) RETURNS varchar AS $$

DECLARE

filmy\_kursor CURSOR FOR SELECT imie, nazwisko from aktorzy natural join filmy where id\_filmu = id;

tytul\_filmu filmy.tytul%type;

rok\_produkcji\_filmu filmy.rok\_produkcji%type;

imie\_f varchar(100);

nazwisko\_f varchar(100);

aktorzy varchar(1000);

wynik varchar(1000);

BEGIN

aktorzy := '';

select tytul, rok\_produkcji into tytul\_filmu, rok\_produkcji\_filmu from filmy where id\_filmu = id;

OPEN filmy\_kursor;

LOOP

FETCH filmy\_kursor INTO imie\_f, nazwisko\_f;

EXIT when not found;

aktorzy := aktorzy || imie\_f || ' ' || nazwisko\_f || ', ';

END LOOP;

CLOSE filmy\_kursor;

wynik :='Film ' || tytul\_filmu || ' zostal nakrecony w roku ' || rok\_produkcji\_filmu || ' przy

udziale aktorow: ' || aktorzy;

wynik := left(wynik, length(wynik)-2);

RETURN wynik || '.' ;

END;

$$ LANGUAGE PLPGSQL;

--- test

select opis\_filmu(1)

--- usunięcie

drop function opis\_filmu

**1. Dla atrybutu ID\_FILMU relacji FILMY zdefiniuj ograniczenie integralnościowe typu UNIQUE, dbające o niepowtarzalność identyfikatorów filmów w obrębie relacji.**

alter table filmy

add unique (id\_filmu)

**2. Utwórz funkcję o nazwie DODAJ\_FILM(id INTEGER, tyt VARCHAR, rok INTEGER, c REAL), służącą do wprowadzania nowej krotki do relacji FILMY. W przypadku, gdy podczas wykonania funkcji wystąpi wyjątek naruszenia ograniczenia integralnościowego UNIQUE, automatycznie zmodyfikuj wartość identyfikatora nowego filmu tak, aby była niepowtarzalna (np. aktualna wartość maksymalna + 1). Przetestuj działanie funkcji. W rozwiązaniu zamieść polecenie tworzące funkcję DODAJ\_FILM**

create or replace function DODAJ\_FILM(id INTEGER, tyt VARCHAR, rok INTEGER, c REAL) RETURNS varchar AS $$

declare

wynik varchar ='sukces';

ostatni integer;

begin

select max(id\_filmu) into ostatni from filmy;

insert into filmy (id\_filmu, tytul, rok\_produkcji, cena) values (id,tyt, rok,c);

return wynik;

exception

when UNIQUE\_VIOLATION then

ostatni=ostatni+1;

wynik = 'byl juz ten id';

insert into filmy (id\_filmu, tytul, rok\_produkcji, cena) values (ostatni,tyt, rok,c);

return wynik;

end;

$$ language PLPGSQL;

select DODAJ\_FILM (10, 'Psi patrol', 2022,67)

select \* from filmy

**3. Usuń funkcję DODAJ\_FILM.**

drop function DODAJ\_FILMY

WYZWALACZE

1. **Utwórz relację AKTORZY\_OPERACJE(data DATE, oper VARCHAR(100)), która posłuży do śledzenia operacji wykonywanych przez użytkownika.**

create table aktorzy\_operacje

(

date date, oper varchar(100)

)

1. **Utwórz wyzwalacz reagujący na operacje wprowadzania i usuwania krotek z relacji AKTORZY. Czas i rodzaj każdej operacji wykonywanej na relacji AKTORZY powinien być zapisywany w relacji AKTORZY\_OPERACJE. Przetestuj działanie wyzwalacza. W rozwiązaniu zamieść wszystkie polecenie potrzebne do utworzenia wyzwalacza.**

CREATE FUNCTION log\_aktorzy() RETURNS TRIGGER AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

insert into aktorzy\_operacje (date, oper)

values

((SELECT CURRENT\_DATE), 'Insert');

ELSIF (TG\_OP = 'DELETE') THEN

insert into aktorzy\_operacje (date, oper)

values

((SELECT CURRENT\_DATE), 'Delete');

END IF;

RAISE NOTICE 'Wyzwalacz uruchomiony - logi zapisane dla modyfikacji!';

RETURN null;

END; $$ LANGUAGE PLPGSQL;

CREATE TRIGGER trigger\_exe after INSERT OR delete ON aktorzy

FOR EACH ROW EXECUTE PROCEDURE log\_aktorzy();

select \* from aktorzy

select \* from aktorzy\_operacje

insert into aktorzy

(id\_aktora, imie, nazwisko)

values

(29, 'Jan', 'Kowalski')

delete from aktorzy

1. where id\_aktora = 29

3. Usuń wyzwalacz i procedurę wyzwalacza.

drop function log\_aktorzy

drop trigger trigger\_exe on aktorzy

**Na potrzeby zadań utwórz tablice pracownicy(id\_pracownika, imie, nazwisko, miasto, pensja)**

create table pracownicy

(

id\_pracownika int primary key,

imie varchar(200),

nazwisko varchar(200),

miasto varchar(200),

pensja real

)

insert into pracownicy

values

(6, 'jan', 'kowalski', 'Komorniki', 85539)

select \* from pracownicy

select pensja from pracownicy

where id\_pracownika = 1

1. **Utwórz w języku PL/pgSQL funkcję PIT(dochód REAL) służącą do wyliczania należnego podatku dochodowego od osób fizycznych według skali podatkowej. Obliczenia powinny odbywać się według następującego algorytmu: - jeżeli "dochód"<= 85528 zł, to podatek należny wynosi 18% z "dochód" minus 556,02 zł - jeżeli "dochód" > 85528 zł, to podatek należny wynosi 14839 zł plus 32% z ("dochód" - 85528 zł) Następnie sprawdź działanie funkcji PIT() na tabeli pracownicy: select nazwisko, pensja, pit(pensja) from pracownicy;**

create or replace function PIT(dochod real) RETURNS real AS $$

declare

podatek real;

podatek\_procent real;

begin

IF (dochod <= 85528) THEN

podatek\_procent := 0.18;

podatek := dochod \* podatek\_procent;

ELSIF (dochod > 85528) THEN

podatek\_procent := 0.32;

podatek := 14839 + dochod \* podatek\_procent;

END IF;

return podatek;

END; $$ LANGUAGE PLPGSQL;

select PIT((select pensja from pracownicy

where id\_pracownika = 6))

select nazwisko, pensja, pit(pensja) from pracownicy;

1. **Utwórz w języku PL/pgSQL funkcję HIRE(id\_pracownika INT, imie VARCHAR, nazwisko VARCHAR, miasto VARCHAR, pensja REAL) służącą do wstawienia nowego rekordu do tabeli PRACOWNICY. Funkcja powinna zwracać wartość tekstową: - "OK", jeżeli wstawianie rekordu odbędzie się bez zgłoszenia wyjątku - "DUPLIKAT ID", jeżeli podczas próby wstawienia rekordu zostanie zgłoszony wyjątek UNIQUE\_VIOLATION Następnie sprawdź działanie funkcji HIRE(): select hire(6, 'Jan', 'Kowalski', 'Poznan', 100); select hire(5, 'Anna', 'Nowak', 'Poznan', 200);**

create or replace function HIRE(id\_pracownika INT, imie VARCHAR, nazwisko VARCHAR, miasto VARCHAR, pensja REAL) RETURNS varchar AS $$

declare

wynik varchar(200) := 'OK';

wynik\_duplat varchar(200) := 'DUPLIKAT ID';

begin

insert into pracownicy values (id\_pracownika, imie, nazwisko, miasto, pensja);

return 'wynik';

EXCEPTION

when unique\_violation then

return wynik\_duplat;

end;

$$ LANGUAGE PLPGSQL;

select hire(7, 'Jan', 'Kowalski', 'Poznan', 100);

select hire(5, 'Anna', 'Nowak', 'Poznan', 200);

**3.Utwórz wyzwalacz DOUBLE\_SALARY i związaną z nim procedurę wyzwalaną DOUBLE\_SALARY\_FUN, który każdemu nowo wstawianemu do tabeli PRACOWNICY rekordowi podwoi wartość kolumny PENSJA. Następnie sprawdź działanie wyzwalacza: insert into pracownicy values(10, 'Jan', 'Kowalski','Poznan', 100);**

**select \* from pracownicy where id\_pracownika=10;**

create or replace function DOUBLE\_SALARY\_FUN() RETURNS TRIGGER AS $$

begin

new.pensja := new.pensja \* 2;

RETURN NEW;

END; $$ LANGUAGE PLPGSQL;

CREATE TRIGGER double\_salary BEFORE INSERT ON pracownicy

FOR EACH ROW EXECUTE PROCEDURE DOUBLE\_SALARY\_FUN();

insert into pracownicy values(11, 'Jan', 'Kowalski', 'Poznan', 100);

select \* from pracownicy where id\_pracownika=11;

drop trigger double\_salary on pracownicy