FUNKCJE

Napisz funkcje o nazwie uzupelnij_playliste która przyjmuje trzy argumenty: idplaylisty_od (int),idplaylisty_do (int), polub (boolean). Funkcja skopiuje z playlisty idplaylisty_od do playlisty idplaylisty_do utwory, które nie występują na tej drugiej. Jeżeli parametr polub jest równy TRUE to dla skopiowanych utworów funkcja doda oceny pozytywne (lubi = TRUE), wystawione przez właściciela drugiej playlisty, ale tylko jeśli jeszcze nie mają od niego ocen. Funkcja zwraca tabelę zawierającą wszystkie utwory (wiersze z tabeli utwory) znajdujące się na playliście idplaylisty_do po operacji kopiowania.

```
CREATE OR REPLACE FUNCTION uzupelnij_playliste(IN idplaylisty_od INTEGER,
IN idplaylisty_do INTEGER, IN polub BOOLEAN)
RETURNS TABLE (
    r_idutworu INTEGER,
    r_idalbumu INTEGER,
    r_nazwa VARCHAR(100),
    r_dlugosc INTEGER
) AS
$$
DECLARE owner INTEGER;
DECLARE c RECORD;
BEGIN
owner := (SELECT idklienta FROM playlisty WHERE idplaylisty =
idplaylisty_do);
FOR c IN SELECT z.idutworu
FROM zawartosc z
WHERE z.idplaylisty = idplaylisty_od
AND NOT EXISTS(SELECT 1 FROM zawartosc zz WHERE zz.idplaylisty =
idplaylisty_do AND zz.idutworu = z.idutworu )
LOOP
INSERT INTO zawartosc VALUES(idplaylisty_do, c.idutworu);
IF polub = TRUE AND NOT EXIST (SELECT * FROM oceny WHERE idklienta = owner
AND idutworu = c.idutworu)
THEN
INSERT INTO oceny (c.idutworu, owner, TRUE);
END IF;
END LOOP;
RETURN QUERY
SELECT DISTINCT idutworu,idalbumu, nazwa, dlugosc
FROM zawartosc z
JOIN utwory u USING(idutworu)
WHERE z.idplaylisty = idplaylisty_do;
END;
$$ LANGUAGE PLpgSQL;
```

1. Napisz funkcje czasTrwania, która jako parametr przyjmuję idplaylisty (int) i zwraca czas trwania danej playlisty

```
-- zapytanie sql zwracające czas trwania playlist --
SELECT DISTINCT z.idplaylisty, SUM(u.dlugosc)
FROM utwory u
JOIN zawartosc z USING(idutworu)
GROUP BY z.idplaylisty ORDER BY z.idplaylisty;
-----------------------------

CREATE OR REPLACE FUNCTION czasTrwania(IN idplay INTEGER)
RETURNS INTEGER AS
$$
DECLARE czas INTEGER;
BEGIN
SELECT SUM(u.dlugosc) INTO czas
FROM utwory u
JOIN zawartosc z USING(idutworu)
WHERE z.idplaylisty = idplay;
RETURN czas;
END;
$$ LANGUAGE PLpgSQl;
```

2. Napisz bezargumentową funkcję max_play która zwróci idplaylisty (int) na której znajduje się najwięcej utworów - jeśli takich playlist jest więcej to zwróć jeden wynik

```
-- zapytanie sql zwraca idplaylist na których znajduje się największa
liczba utworów --
WITH x AS (
    SELECT idplaylisty, DENSE_RANK() OVER (ORDER BY COUNT(idutworu) DESC)
AS rank
    FROM zawartosc
    GROUP BY idplaylisty
)
SELECT idplaylisty
FROM x
WHERE rank = 1;
---------------------------------
CREATE OR REPLACE FUNCTION max_play()
RETURNS INTEGER AS
$$
DECLARE id INTEGER;
BEGIN
SELECT idplaylisty INTO id
FROM zawartosc
GROUP BY idplaylisty
ORDER BY COUNT(idutworu) DESC
LIMIT 1;
RETURN id;
END;
$$ LANGUAGE PLpgSQL;
```

3. Napisz bezargumentową funkcję min_play która zwróci idplaylist (TABLE) na których znajduje się najmniej utworów - playlisty mogą być puste, wtedy przyjmij że jest na nich 0 utworów

```
----Zapytanie SQL zwracające idplaylist na których znajduje się najmniej
utworów ---
WITH x AS (
    SELECT p.idplaylisty, DENSE_RANK() OVER (ORDER BY
COALESCE(COUNT(z.idutworu), 0) ASC) AS rank
    FROM playlisty p
    LEFT JOIN zawartosc z USING(idplaylisty)
    GROUP BY p.idplaylisty
)
SELECT idplaylisty
FROM x
WHERE rank = 1;
-----------------------------
CREATE OR REPLACE FUNCTION min_play()
RETURNS TABLE(idplaylist INTEGER) AS
$$
DECLARE min_count INTEGER;
BEGIN
SELECT MIN(liczba_utworow) INTO min_count
FROM (
    SELECT p.idplaylisty, COALESCE(COUNT(z.idutworu), 0) AS liczba_utworow
    FROM playlisty p
    LEFT JOIN zawartosc z USING(idplalisty)
    GROUP BY p.idplaylisty
) AS counts;
RETURN QUERY
SELECT idplaylisty
FROM (
    SELECT p.idplaylisty, COALESCE(COUNT(z.idutworu), 0) AS liczba_utworow
    FROM playlisty p
    LEFT JOIN zawartosc z USING(idplalisty)
    GROUP BY p.idplaylisty
) AS counts
WHERE liczba utworow = min_count;
RETURN;
END;
$$ LANGUAGE PLpgSQL;
```

4. Napisz funkcje utwory która przyjmuje nazwę playlisty (VaRCHAR(30)) i zwraca listę utworów (ich nazwy) które się na niej znajdują

```
CREATE OR REPLACE FUNCTION utwory(IN arg1 VARCHAR(30))
RETURNS TABLE( r_utwor VARCHAR(100)) AS
$$
DECLARE id INTEGER;
BEGIN
id := (SELECT idplaylisty FROM playlisty WHERE nazwa = arg1);
RETURN QUERY
SELECT u.nazwa
```

```
FROM playlisty p
JOIN zawartosc z USING(idplaylisty)
JOIN utwory u USING(idutworu)
WHERE idplaylisty = id;
RETURN;
END;
$$ LANGUAGE PLpgSQL;
```

5. Napisz funkcje playlisty która przyjmuję nazwę utworu i zwraca liczbę playlist na której dany utwór się znajduje - jeśli nie znajduje się na żadnej funkcja ma zwrócić 0

```
--- Zapytanie SQL liczące na ilu playlistach znajduje się dany utwór ----
SELECT u.idutworu, COALESCE(COUNT(z.idplaylisty),0)
FROM utwory u
LEFT JOIN zawartosc z USING(idutworu)
GROUP BY u.idutworu;
-------------------------------------------------
CREATE OR REPLACE FUNCTION playlisty(IN arg1 VARCHAR(100))
RETURNS INTEGER AS
$$
DECLARE liczba INTEGER;
BEGIN
SELECT COALESCE(COUNT(z.idplaylisty),0) INTO liczba
FROM utwory u
LEFT JOIN zawartosc z USING(idutworu)
WHERE u.nazwa = arg1
GROUP BY u.idutworu;
RETURN liczba;
END;
$$ LANGUAGE PLpgSQL;
```

6. Napisz funkcje puste_playlisty która zwraca listę playlist(ich id) na których nie znajdują się żadne utwory

```
--------- zapytanie sql zwracające playlisty na których nie znajdują się
żadne utwory -----
SELECT p.idplaylisty
FROM playlisty p
LEFT JOIN zawartosc z USING(idplaylisty)
WHERE z.idutworu IS NULL;
--------------------------------------
CREATE OR REPLACE FUNCTION puste_playlisty()
RETURNS TABLE(r_idplaylisty INTEGER) AS
$$
BEGIN
RETURN QUERY
SELECT p.idplaylisty
FROM playlisty p
LEFT JOIN zawartosc z USING(idplaylisty)
WHERE z.idutworu IS NULL;
RETURN;
END;
$$ LANGUAGE PLpgSQL;
```

7. Napisz funkcje utwory_od_do przyjmującą trzy argumenty: idplaylisty, czas_od, czas_do zwracającą wszystkie utwory(ich id i nazwę) na podanej playliście których czas trwania mieści się w zadanych granicach

```
----- zapytanie sql zwracające utwory mieszczące sie w podanych granicach
czasowych na danej playliście---
SELECT z.idutworu
FROM zawartosc z
JOIN utwory u USING(idutworu)
WHERE z.idplaylisty = XYZ
AND u.dlugosc BETWEEEN czas_od AND czas_do;
-------------------------------------------
CREATE OR REPLACE FUNCTION utwory_od_do(IN idplay INTEGER, IN czas_od
INTEGER, IN czas_do INTEGER)
RETURNS TABLE(
    r_idutworu INTEGER,
    r_nazwa VARCHAR(100)
) AS
$$
BEGIN
RETURN QUERY
SELECT u.idutworu, u.nazwa
FROM utwory u
JOIN zawartosc z USING(idutworu)
WHERE z.idplaylisty = idplay
AND u.dlugosc BETWEEN czas_od AND czas_do;
RETURN;
END;
$$ LANGUAGE PLpgSQL;
```

8. Napisz funkcje dodaj_utwor, która przyjmuje argumenty: idutworu, nazwę utworu, idalbumu, dlugosc utworu oraz nazwę playlisty na której znajdzie się nowo dodany utwór. Funkcja zwraca wszystkie utwory(ich nazwy) na zaktualizowanej playliście, których długość jest co najmniej równa długości nowo dodanego utworu.

```
---- pomocnicze/ koncepcyjne zapytania sql ----
INSERT INTO utwory(idutworu, nazwa, dlugosc) VALUES(id, name, time);
INSERT INTO zawartosc(idutworu, idplaylisty) VALUES(id_u, id_p);
SELECT u.nazwa
FROM utwory u
JOIN zawartosc z USING(idutworu)
WHERE z.idplaylisty = XYZ
AND u.dlugosc >= czas;
-----------------------------------
CREATE OR REPLACE FUNCTION dodaj_utwor(IN id_u INTEGER, IN nazwa_u
VARCHAR(100),IN id_a INTEGER,IN dlugosc_u INTEGER, IN nazwa_p VARCHAR(30))
RETURNS TABLE(
    r_nazwa_u VARCHAR(100)
) AS
```

```
$$
DECLARE id_playlisty INTEGER;
BEGIN
id_playlisty := (SELECT idplaylisty FROM playlisty WHERE nazwa = nazwa_p);
INSERT INTO utwory(idutworu, nazwa, idalbumu, dlugosc) VALUES(id_u,
nazwa_u,id_a, dlugosc_u);
INSERT INTO zawartosc(idutworu, idplaylisty) VALUES(id_u, id_playlisty);
RETURN QUERY
SELECT u.nazwa
FROM utwory u
JOIN zawartosc z USING(idutworu)
WHERE z.idplaylisty = id_playlisty
AND u.dlugosc >= dlugosc_u;
RETURN;
END;
$$ LANGUAGE PLpgSQL;
```

9. Napisz funkcje klienci która przyjmuje loginklienta_od, funkcja zwraca id wszystkich klientów którzy mają na swoich playlistach co najmniej jeden utwór pokrywający się z utworami na playliście klienta o loginie loginklienta_od i którzy dodatkowo urodzili się po tym jak klient_od się zarejestrował.

```
---- pomocnicze/ koncepcyjne zapytania sql----
SELECT p.idklienta
FROM playlisty p
JOIN zawartosc z USING(idplaylisty)
JOIN klienci kk USING(idklienta)
WHERE z.idutworu IN (SELECT idutworu FROM zawartosc JOIN playlisty pp
USING(idplaylisty
JOIN klienci k USING(idklienta)
WHERE k.nazwa = (SELECT nazwa FROM klienci WHERE
login='login_klienta_od')))
AND kk.data_urodzenia > (SELECT data_rejestracji FROM klienci WHERE login =
'login klienta od');
-----------------------------------------------
CREATE OR REPLACE FUNCTION klienci(IN loginKlienta_od VARCHAR(100))
RETURNS TABLE(
    r_idklienta INTEGER
) AS
$$
DECLARE id_klienta_od INTEGER;
BEGIN
id_klienta_od := (SELECT idklienta FROM klienci WHERE login =
'loginKlienta_od');
RETURN QUERY
SELECT k.idklienta
FROM klienci k
JOIN playlisty p USING(idklienta)
JOIN zawartosc z USING(idplaylisty)
WHERE z.idutworu IN (SELECT zz.idutworu FROM zawartosc zz JOIN playlisty pp
USING(idplaylisty) WHERE idklienta = id_klienta_od)
AND k.data_urodzenia > (SELECT data_rejestracji FROM klienci WHERE
idklienta = id_klienta_od);
RETURN;
```

```
END;
$$ LANGUAGE PLpgSQL;
```

10. Napisz funkcje start_od która przyjmuje prefiks i zwraca dane utworów (idutworu, idalbumu, nazwa, dlugosc), które się zaczynają od prefiksu

```
CREATE OR REPLACE FUNCTION start_od(IN arg1 CHAR(10))
RETURNS TABLE(
    r_idutworu INTEGER,
    r_idalbumu INTEGER,
    r_nazwa VARCHAR(100),
    r_dlugosc INTEGER
) AS
$$
BEGIN
RETURN QUERY
SELECT idutworu, idalbumu, nazwa, dlugosc
FROM utwory
WHERE nazwa ~('^' || arg1) ;
RETURN;
END;
$$ LANGUAGE PLpgSQL;
```

11. Napisz funkcje kopiuj_avg która przyjmuje dwa argumenty : nazwaplaylisty_od, nazwaplaylisty_do, która skopiuje wszystkie utwory które nie występują na playliście_do, z playlisty_od do playlisty_do, jeśli czas ich trwania jest większy niż średni czas trwania utworów (INTEGER) na playliście_do, Funkcja zwraca wszystkie utwory na playliście_do po procesie kopiowania (idutworu, idalbumu, nazwa, dlugosc)

```
---- zapytania pomocnicze ---
--- średni czas trwannia utworu na playliście ---
SELECT DISTINCT p.idplaylisty, COALESCE(AVG(u.dlugosc), 0)
FROM playlisty p
LEFT JOIN zawartosc z USING(idplaylisty)
LEFT JOIN utwory u USING(idutworu)
GROUP BY p.idplaylisty;
-------------------------------------
CREATE OR REPLACE FUNCTION kopiuj_avg(IN nazwa_playlisty_od VARCHAR(30), IN
nazwa_playlisty_do VARCHAR(30))
RETURNS TABLE(
    r_idutworu INTEGER,
    r_idalbumu INTEGER,
    r_nazwa VARCHAR(100),
    r_dlugosc INTEGER
) AS
$$
DECLARE idp_od INTEGER;
```

```
DECLARE idp_do INTEGER;
DECLARE c RECORD;
DECLARE avg INTEGER;
BEGIN
idp_od := (SELECT idplaylisty FROM playlisty WHERE nazwa =
nazwa_playlisty_od);
idp_do := (SELECT idplaylisty FROM playlisty WHERE nazwa =
nazwa_playlisty_do);
avg := (SELECT COALESCE((AVG(u.dlugosc),0))::INTEGER FROM playlisty p JOIN
zawartosc z USING(idplaylisty) JOIN utwory u USING(idutworu) WHERE
p.idplaylisty = idp_do);

FOR c IN SELECT z.idutworu FROM zawartosc z JOIN utwory u USING(idutworu)
WHERE z.idplaylisty = idp_od
AND NOT EXISTS (
            SELECT 1
            FROM zawartosc zz
            WHERE zz.idplaylisty = idp_do
            AND zz.idutworu = z.idutworu
        )
AND u.dlugosc > avg
LOOP
INSERT INTO zawartosc VALUES(idp_do, c.idutworu);
END LOOP;
RETURN QUERY
SELECT u.idutworu, u.idalbumu, u.nazwa, u.dlugosc
FROM zawartosc z
JOIN utwory u USING(idutworu)
WHERE z.idplaylisty = idp_do;
RETURN;
END;
$$ LANGUAGE PLpgSQL;
```

12. Napisz funkcję kopiuj_zaczynajace_sie_od która przyjmuje 3 argumenty : idplaylisty_od, nazwaplaylisty_do, prefiks (varchar(10)). Funkcja kopiuje wszystkie utowry, których nazwa zaczyna się od prefiks i które nie występują na playliście_do, z playlisty_od do playlisty_do.Funkcja zwraca wszystkie utwory na playliście_do po procesie kopiowania (idutworu, idalbumu, nazwa, dlugosc)

```
CREATE OR REPLACE FUNCTION kopiuj_zaczynajace_sie_od(IN idplaylisty_od
INTEGER, IN nazwaplaylisty_do VARCHAR(30), IN prefiks VARCHAR(10))
RETURNS TABLE (
    r_idutworu INTEGER,
    r_idalbumu INTEGER,
    r_nazwa VARCHAR(100),
    r_dlugosc INTEGER
) AS
$$
DECLARE idp_do INTEGER;
DECLARE c RECORD;
BEGIN
```

```
idp_do := (SELECT idplaylisty FROM playlisty WHERE nazwa =
nazwaplaylisty_do);
FOR c IN SELECT u.idutworu FROM zawartosc z JOIN utwory u USING(idutworu)
WHERE z.idplaylisty = idplaylisty_od AND u.nazwa ~('^' || prefiks)
AND NOT EXISTS(
    SELECT 1
    FROM zawartosc zz
    WHERE z.idplaylisty = idp_do
    AND zz.idutworu = z.idutworu
)
LOOP
INSERT INTO zawartosc VALUES(idp_do, c.idutworu);
END LOOP;
RETURN QUERY
SELECT idutworu, idalbumu, nazwa, dlugosc
FROM utwory
JOIN zawartosc USING(idutworu)
WHERE idplaylisty = idp_do;
END;
$$ LANGUAGE PLpgSQL;
```

13. Napisz funkcje dodaj_utwory_wykonawcy1 która przyjmuje 2 argumenty: nazwa_wykonawcy, login_klienta. Funkcja dodaje do każdej playlisty nazleżącej do klienta o loginie login_klienta utwory wytępujące na albumach wykonawcy nazwa_wykonawcy - o ile wcześniej nie znajdują się już na jego playliście. Album z którego pochodzą utwory musi być dodatkowo wydany przed datą rejestracji klienta. Funkcja zwraca idplaylist oraz wszystkie utwory na playlistach klienta po dodaniu do nich utworów (idutworu, idalbumu, nazwa, dlugosc).

```
CREATE OR REPLACE FUNCTION dodaj_utwory_wykonawcy1(IN nazwa_wykonawcy
VARCHAR(100), IN login_klienta VARCHAR(50))
RETURNS TABLE(
    r_idplaylisty INTEGER,
    r_idutworu INTEGER,
    r_idalbumu INTEGER,
    r_nazwa VARCHAR(100),
    r_dlugosc INTEGER
) AS
$$
DECLARE id_k INTEGER;
DECLARE id_w INTEGER;
DECLARE c1 RECORD;
DECLARE c2 RECORD;
BEGIN
id_k := (SELECT idklienta FROM klienci WHERE login = login_klienta);
id_w := (SELECT idwykonawcy FROM wykonawcy WHERE nazwa = nazwa_wykonawcy);
FOR c1 IN SELECT DISTINCT idplaylisty
FROM  playlisty p
```

```
WHERE p.idklienta = id_k
LOOP
FOR c2 IN SELECT u.idutworu
FROM utwory u
JOIN albumy a USING(idalbumu)
WHERE a.idwykonawcy = id_w
AND NOT EXISTS(SELECT 1 FROM zawartosc zz WHERE zz.idplaylisty =
c1.idplaylisty AND u.idutworu = zz.idutworu)
AND a.data_wydania < (SELECT data_rejestracji FROM klienci WHERE idklienta
= id_k)
LOOP
INSERT INTO zawartosc VALUES(c1.idplaylisty, c2.idutworu);
END LOOP;
END LOOP;
RETURN QUERY
SELECT p.idplaylisty, u.idutworu, u.idalbumu, u.nazwa, u.dlugosc
FROM utwory u
JOIN zawartosc z USING(idutworu)
JOIN playlisty p USING(idplaylisty)
WHERE idklienta = id_k;
END;
$$ LANGUAGE PLpgSQL;
```

13. Napisz funkcje dodaj_utwory_wykonawcy2 która przyjmuje 3 argumenty: idwykonawcy, idklienta, prefiks(VARCHAR(10)). Funkcja dodaje do każdej playlisty nazleżącej do klienta o id = idklienta, utwory zaczynające się od prefiks wytępujące na albumach wykonawcy o id idwykonawcy - o ile wcześniej nie znajdują się już na jego playliście.Funkcja zwraca wszystkie utwory na playlistach klienta po dodaniu do nich utworów (idutworu, idalbumu, nazwa, dlugosc).

```
CREATE OR REPLACE FUNCTION dodaj_utwory_wykonawcy2(IN id_w INTEGER, IN id_k
INTEGER, IN prefiks VARCHAR(10))
RETURNS TABLE (
    r_idutworu INTEGER,
    r_idalbumu INTEGER,
    r_nazwa VARCHAR(100),
    r_dlugosc INTEGER
) AS
$$
DECLARE c1 RECORD;
DECLARE c2 RECORD;
BEGIN
FOR c1 IN SELECT idplaylisty FROM playlisty p WHERE p.idklienta = id_k
LOOP
FOR c2 IN SELECT u.idutworu FROM utwory u JOIN albumy a USING(idalbumu)
WHERE a.idwykonawcy = id_w AND u.nazwa ~('^' || prefiks)
AND NOT EXISTS(SELECT 1 FROM zawartosc zz WHERE zz.idplaylisty =
c1.idplaylisty AND zz.idutworu = u.idutworu)
LOOP
INSERT INTO zawartosc VALUES(c1.idplaylisty, c2.idutworu);
```

```
END LOOP ;
END LOOP;
RETURN QUERY
SELECT u.idutworu, u.idalbumu, u.nazwa, u.dlugosc
FROM utwory u
JOIN zawartosc z USING(idutworu)
JOIN playlisty p USING(idplaylisty)
WHERE p.idklienta = id_k;
END;
$$ LANGUAGE PLpgSQL;
```

14. Napisz funkcję liczba_z_kraju przyjmującą idplaylisty oraz kraj. Funkcja zwraca liczbę utworów znajdujących się na playliście które pochodzą z albumów wydanych w danym kraju.

```
CREATE OR REPLACE FUNCTION liczba_z_kraju(IN id_p INTEGER, IN k
VARCHAR(30))
RETURNS INTEGER AS
$$
DECLARE c INTEGER;
BEGIN
SELECT COUNT(z.idutworu) INTO c
FROM zawartosc z
JOIN utwory u USING(idutworu)
JOIN albumy a USING(idalbumu)
JOIN wykonawcy w USING(idwykonawcy)
WHERE z.idplaylisty = id_p
AND w.kraj = k;
RETURN c;
END;
$$ LANGUAGE PLpgSQL;
```

15. Napisz funkcje polub_utwory_wykonawcy która przyjmuje 3 argumenty: loginklienta, idwykonawcy, polub(boolean). Funkcja dodaje polubienia do wszystkich utworów znajdujących się na wszystkich playlistach klienta o loginie loginklienta (jeśli wartość polub jest ustawiona na True), ktore należą do albumów wykonawcy o id = idwykonawcy i nie zostały wcześniej polubione przez

klienta. Funkcja zwraca wszystkie polubione utwory z playlist klienta (po operacji polubienia utworów wykonawców) (idplaylisty, idutworu, idklienta, lubi)

```
CREATE OR REPLACE FUNCTION  polub_utwory_wykonawcy(IN l_k VARCHAR(50), IN
id_w INTEGER, IN polub BOOLEAN)
RETURNS TABLE(
    r_idplaylisty INTEGER,
    r_idutworu INTEGER,
    r_idklienta INTEGER,
    r_lubi BOOLEAN
) AS
$$
DECLARE id_k INTEGER;
DECLARE c1 RECORD;
DECLARE c2 RECORD;
BEGIN
id_k := (SELECT idklienta FROM klienci WHERE login = l_k);
FOR c1 IN SELECT idplaylisty FROM playlisty WHERE idklienta = id_k
LOOP
FOR c2 IN SELECT DISTINCT u.idutworu FROM utwory u JOIN albumy a
USING(idalbumu) JOIN zawartosc p USING(idutworu) JOIN oceny o
USING(idutworu)
WHERE p.idplaylisty = c1.idplaylisty AND a.idwykonawcy = id_w
AND NOT EXISTS(SELECT 1 FROM oceny WHERE idutworu = u.idutworu AND lubi =
TRUE)
LOOP
IF polub = TRUE THEN
INSERT INTO oceny VALUES(c2.idutworu, id_k, TRUE);
END IF;
END LOOP ;
END LOOP ;
RETURN QUERY
SELECT z.idplaylisty, o.idutworu, o.idklienta, o.lubi
FROM zawartosc z
JOIN utwory u USING(idutworu)
JOIN oceny o USING(idutworu)
WHERE o.idklienta = id_k;
END;
$$ LANGUAGE PLpgSQL;
```

16. Napisz bezargumentową funkcję wydluz_trwanie, która przedłuża czas trwania utworów pochodzących z albumów wykonawców którzy już zakończyli karierę o: 20s dla utworów trwających poniżej 200 s, 25 s dla utworów trwających od 200 do 300 s, 30 s dla pozostałych utworów. Funkcja zwraca listę utworów (nazwy) wraz z nowym czasem ich trwania.

```
CREATE OR REPLACE FUNCTION wydluz_trwanie()
RETURNS TABLE(
    r_nazwa_u VARCHAR(100),
    r_dlugosc_u INTEGER
) AS
$$
DECLARE c RECORD;
DECLARE z INTEGER;
BEGIN
FOR c IN SELECT * FROM utwory u JOIN albumy a USING(idalbumu) JOIN
wykonawcy w USING(idwykonawcy) WHERE w.data_zakonczenia IS NOT NULL
LOOP
IF c.dlugosc < 200 THEN
z := 20;
ELSIF c.dlugosc BETWEEN 200 AND 300 THEN
z := 25;
ELSE
z:=30;
END IF;
UPDATE utwory
SET dlugosc = dlugosc + z WHERE idutworu = c.idutworu;
END LOOP ;
RETURN QUERY
SELECT u.nazwa, u.dlugosc
FROM utwory u
JOIN albumy a USING(idalbumu)
JOIN wykonawcy w USING(idwykonawcy)
WHERE w.data_zakonczenia IS NOT NULL;
END;
$$ LANGUAGE PLpgSQL;
```

17. Napisz bezargumentową funkcje skroc_czas_trwania, która skraca czas trwania utworów ktore znajdują się na albumach wydanych przez wykonawców z United Kingdom którzy jeszcze nie zakończyli kariery o : 20 s dla utworów trwających pomiędzy 200 a 300 s, 30 dla utworów trwających powyżej 300 s, 10 dla utworów poniżej 200 s. Funkcja zwraca nazwy utworów, idalbumu oraz czas trwania utworu.

```
CREATE OR REPLACE FUNCTION skroc_czas_trwania()
RETURNS TABLE(
    r_u_nazwa VARCHAR(100),
    r_idalbumu INTEGER,
    r_dlugosc INTEGER
) AS
$$
DECLARE z INTEGER;
DECLARE c RECORD;
BEGIN
FOR c IN SELECT * FROM utwory u JOIN albumy a USING(idalbumu) JOIN
wykonawcy w USING(idwykonawcy) WHERE w.kraj = 'United Kingdom' AND
w.data_zakonczenia IS NULL
LOOP
```

```
        IF c.dlugosc < 200 THEN
            z := 10;
        ELSIF c.dlugosc BETWEEN 200 AND 300 THEN
            z := 20;
        ELSE
            z := 30;
        END IF;

UPDATE utwory
SET dlugosc=dlugosc - z
WHERE idutworu = c.idutworu;
END LOOP;
RETURN QUERY
SELECT u.nazwa, u.idalbumu, u.dlugosc
FROM utwory u
JOIN albumy a USING(idalbumu)
JOIN wykonawcy w USING(idwykonawcy)
WHERE w.kraj = 'United Kingdom'
AND w.data_zakonczenia IS NULL;
END;
$$ LANGUAGE PLpgSQL;
```

18. Napisz funkcję dodajUtwor, która jako argument przyjmuje idalbumu i zwraca void. Funkcja dodaje nowy utwór do albumu o id = idalbumu. Długość utworu jest taka sama jak średnia długość utworów już wcześniej znajdujących się na danym albumie. Jako idUtworu należy podać wartość 200 jako nazwę - test 200.

```
CREATE OR REPLACE FUNCTION dodajUtwor(IN a_idalbumu INTEGER)
RETURNS VOID AS
$$
DECLARE avg_u INTEGER;
BEGIN
avg_u = (SELECT AVG(u.dlugosc)::INTEGER FROM albumy a JOIN utwory u
USING(idalbumu) WHERE idalbumu = a_idalbumu);
INSERT INTO utwory VALUES(200, a_idalbumu, 'test200', avg_u);
END;
$$ LANGUAGE PLpgSQL;
```

19. Napisz funkcje dodajOcene, która jako argumenty przyjmuje idalbumu i idklienta i zwraca void. Funkcja dla wszystkich utworów obecnych na albumie o id = idalbumu dodaje oceny pozytywne jeśli średnia ich ocen jest >= 0.5, w przeciwnym wypadku dodaje oceny negatywne(jeśli dany utwór nie ma żadnej oceny to przyjmij że średnia jego ocen = 0).

```
CREATE OR REPLACE FUNCTION dodajOcene(IN a_idalbumu INTEGER, IN id_k
INTEGER)
```

```
RETURNS VOID AS
$$
DECLARE c RECORD;
DECLARE avg NUMERIC(7, 2);
BEGIN
FOR c IN SELECT DISTINCT u.idutworu FROM utwory u JOIN albumy a
USING(idalbumu) WHERE idalbumu = a_idalbumu
LOOP
avg := (SELECT COALESCE(AVG(o.lubi::INT), 0) FROM utwory u LEFT JOIN oceny
o USING(idutworu) WHERE o.idutworu = c.idutworu);
IF avg >= 0.5 THEN
INSERT INTO oceny VALUES(c.idutworu, id_k, TRUE);
ELSE
INSERT INTO oceny VALUES(c.idutworu, id_k, FALSE);
END IF;
END LOOP;
END;
$$ LANGUAGE PLpgSQL;
```

20. Napisz funkcje utworyNaPlaylistach ktora przyjmuje idklienta i zwraca listę utworów(idutworu, idalbumu, nazwa, dlugosc) na playlistach klienta wraz z łączną liczbą ocen każdego z nich.

```
----------utwory na playliście klienta i id = id_klienta---------
SELECT z.idutworu
FROM zawartosc z
WHERE z.idplaylisty = (SELECT idplaylisty FROM playlisty WHERE idklienta =
id_klienta);

------łączna liczba ocen dla utworów na playilstach klienta, utwór może nie
mieć żadnych ocen -------------
SELECT u.idutworu, COALESCE(COUNT(o.idutworu), 0)
FROM utwory u
LEFT JOIN oceny o USING(idutworu)
JOIN zawartosc z USING(idutworu)
WHERE z.idplaylisty = (SELECT idplaylisty FROM playlisty WHERE idklienta =
id_klienta)
GROUP BY u.idutworu;
--------- funkcja -----
CREATE OR REPLACE FUNCTION utworyNaPlaylistach(IN id_k INTEGER)
RETURNS TABLE (
    r_idutworu INTEGER,
    r_idalbumu INTEGER,
    r_nazwa VARCHAR(100),
    r_dlugosc INTEGER,
    r_liczba INTEGER
) AS
$$
BEGIN
RETURN QUERY
SELECT u.idutworu, u.idalbumu, u.nazwa, u.dlugosc,
COALESCE(COUNT(o.idutworu), 0)::INT
FROM utwory u
LEFT JOIN oceny o USING(idutworu)
JOIN zawartosc z USING(idutworu)
WHERE z.idplaylisty = (SELECT idplaylisty FROM playlisty WHERE idklienta =
id_k)
```

```
GROUP BY u.idutworu;
END;
$$ LANGUAGE PLpgSQL;
```

21. Napisz funkcje korektaDługosci która przyjmuje identyfikator klienta (idklienta) która zmienia aktualną wartość długości utworów występujących na playlistach podanego klienta na wartość średnią długości wszystkich utworów znajdujących się na playlistach klientów zarejestrowanych po zarejestrowaniu się podanego klienta. Funkcja zwraca listę zaktualizowanych utworów wraz z ich nową długością oraz playlistą na której znajduje się dany utwór (idutworu, nazwa, dlugosc, idplaylisty)

```
-------- obliczenie wartości średniej wszystkich utworów znajdujących się
na playlistach klientów zarejestrowanych po urodzeniu podanego klienta-----
---identyfikacja klientów zarejestrowanych po urodzeniu się podanego
klienta ----
SELECT idklienta
FROM klienci
WHERE data_rejestracji > (SELECT data_rejestracji FROM klienci WHERE
idklienta = id_klienta);
--- obliczenie średniej długości utworów na wszystkich playlistach tych
klientów -----
SELECT AVG(u.dlugosc)::INT
FROM utwory u
JOIN zawartosc z USING(idutworu)
WHERE z.idplaylisty IN (SELECT idplaylisty FROM playlisty WHERE idklienta
IN (SELECT idklienta
FROM klienci
WHERE data_rejestracji > (SELECT data_rejestracji FROM klienci WHERE
idklienta = id_klienta)));
-------------Funkcja-------------------------
CREATE OR REPLACE FUNCTION korektaDlugosci(IN id_k INTEGER)
RETURNS TABLE (
    r_idutworu INTEGER,
    r_nazwa VARCHAR(100),
    r_dlugosc INTEGER,
    r_idplaylisty INTEGER
) AS
$$
DECLARE avg INTEGER;
DECLARE c1 RECORD;
DECLARE c2 RECORD;
BEGIN
avg := (SELECT AVG(u.dlugosc)::INT
FROM utwory u
JOIN zawartosc z USING(idutworu)
WHERE z.idplaylisty IN (SELECT idplaylisty FROM playlisty WHERE idklienta
IN (SELECT idklienta
FROM klienci
WHERE data_rejestracji > (SELECT data_rejestracji FROM klienci WHERE
idklienta = id_k))));
```

```
FOR c1 IN SELECT idplaylisty FROM playlisty WHERE idklienta = id_k
LOOP
FOR c2 IN SELECT z.idutworu FROM zawartosc z WHERE z.idplaylisty =
c1.idplaylisty
LOOP
UPDATE utwory
SET dlugosc = avg
WHERE idutworu = c2.idutworu;
END LOOP;
END LOOP;
RETURN QUERY
SELECT u.idutworu, u.nazwa, u.dlugosc, z.idplaylisty
FROM utwory u
JOIN zawartosc z USING(idutworu)
WHERE z.idplaylisty IN (SELECT idplaylisty FROM playlisty WHERE idklienta =
id_k);
END;
$$ LANGUAGE PLpgSQL;
```

22. Napisz funkcje korektaDlugosciUtworu która jako argument przyjmuje idutworu. Jeśli utwór był częściej oceniany(NUMERIC(4, 2)) niż średnia dla utworów (NUMERIC(4,2))to funkcja powinna podnieść jego długość o 10%, natomiast jeżeli rzadziej to obniżyć jego długość o 10%. Funkcja zwraca nazwę utworu i jego długość po korekcie.

```
------- obliczanie średniej częstotliwości oceniania utworów -----
SELECT (SUM(x.liczba_ocen) / COUNT(x.liczba_u) )::NUMERIC(4,2) AS srednia
FROM(
SELECT COALESCE(COUNT(o.idutworu), 0) AS liczba_ocen, COUNT(u.idutworu) AS
liczba_u
FROM utwory u
LEFT JOIN oceny o USING(idutworu)
GROUP BY u.idutworu) x;
--------- obliczanie ile ocen ma dany utwór --
SELECT COALESCE(COUNT(o.idutworu), 0)
FROM utwory u
LEFT JOIN oceny o USING(idutworu)
WHERE u.idutworu = id_u;
------ funkcja ----
CREATE OR REPLACE FUNCTION korektaDlugosciUtworu(IN id_u INTEGER)
RETURNS TABLE (
    r_nazwa_u VARCHAR(100),
    r_dlugosc_u INTEGER
) AS
$$
DECLARE dlugosc_po_korekcie INTEGER;
DECLARE srednia_czestotliwosc NUMERIC(4, 2);
DECLARE czestotliwosc_u NUMERIC(4, 2);
DECLARE zmiana INTEGER;
DECLARE dlugosc_u INTEGER;
BEGIN
dlugosc_u := (SELECT dlugosc FROM utwory WHERE idutworu = id_u);
```

```
zmiana := dlugosc_u / 10;

srednia_czestotliwosc := (SELECT (SUM(x.liczba_ocen) / COUNT(x.liczba_u)
)::NUMERIC(4,2)
FROM(
SELECT COALESCE(COUNT(o.idutworu), 0) AS liczba_ocen, COUNT(u.idutworu) AS
liczba_u
FROM utwory u
LEFT JOIN oceny o USING(idutworu)
GROUP BY u.idutworu) x);

czestotliwosc_u := (SELECT COALESCE(COUNT(o.idutworu), 0)
FROM utwory u
LEFT JOIN oceny o USING(idutworu)
WHERE u.idutworu = id_u);

IF czestotliwosc_u > srednia_czestotliwosc THEN
dlugosc_po_korekcie := dlugosc_u + zmiana;
ELSE
dlugosc_po_korekcie := dlugosc_u - zmiana;
END IF;

UPDATE utwory
SET dlugosc = dlugosc_po_korekcie
WHERE idutworu = id_u;

RETURN QUERY
SELECT nazwa, dlugosc
FROM utwory
WHERE idutworu = id_u;
END;
$$ LANGUAGE PLpgSQL;
```

23. Baza danych z egzaminu 2013-14 termin 2 pizzeria. Zadanie 1. Napisz bezargumentową funkcję podwyzka, która dokonuje podwyżki kosztów składników o : 10% dla składników których koszt jest mniejszy od 2.50 zł, 15% dla składników których koszt jest z przedziału 2.51 zł - 4,00 zł, 18% dla pozostałych. Funkcja powinna ponadto podnieść cenę pizzy o tyle o ile zmienił się łącznie koszt jej składników

```
CREATE OR REPLACE FUNCTION podwyzka()
RETURNS VOID AS
$$
DECLARE nowy_koszt NUMERIC(7, 2);
DECLARE zmiana NUMERIC(7,2);
DECLARE c RECORD;
DECLARE c1 RECORD;
BEGIN
FOR c IN SELECT * FROM skladniki
LOOP
IF c.koszt < 2.50 THEN
zmiana := c.koszt * 10/100;
```

```
ELSIF c.koszt BETWEEN 2.51 AND 4.00 THEN
zmiana:= c.koszt * 15/100;
ELSE
zmiana := c.koszt * 18/100;
END IF;

nowy_koszt := c.koszt + zmiana;

UPDATE skladniki
SET koszt = nowy_koszt
WHERE idskladnika = c.idskladnika;

FOR c1 IN SELECT * FROM  zawartosc  WHERE idskladnika = c.idskladnika
LOOP
UPDATE pizze
SET cena = cena + (zmiana * c1.sztuk) WHERE idpizzy = c1.idpizzy;

END LOOP;
END LOOP;
END:
$$ LANGUAGE PLpgSQL;
```

24. Baza danych z kolokwium 2018/19 - rzeki. Grupa B zad 5. Napisz funkcje dodajOstrzezenie, ktora jako argument przyjmuje ID pomiaru i zwraca void. Funkcja dodaje nowe ostrzeżenie dla podanego pomiaru. Czas ostrzeżenia jest taki sam jak czas pomiaru. W nowym rekordzie należy zignorować kolumnę zmiana_poziomu (nie podawać w zapytaniu INSERT). Jako id_ostrzeżenia należy podać wartość 1963. Jako przekroczony_stan_alarm należy podać wartość null. Należy obliczyć właściwe przekroczenie stanu ostrzegawczego zgodnie ze wzorem: ostrzezenia.przekroczony_stan_ostrz = pomiary.poziom_wody - punkty_pomiarowe.stan_ostrzegawczy.

```
CREATE OR REPLACE FUNCTION dodajOstrzezenie(IN id_p INTEGER)
RETURNS VOID AS
$$
DECLARE przekroczenie_stanu INTEGER;
DECLARE czas_ostrz TIMESTAMP;
DECLARE idpunktu INTEGER;
DECLARE s_o INTEGER;
DECLARE poziomwody INTEGER;
BEGIN
czas_ostrz := (SELECT czas_pomiaru FROM pomiary WHERE idpomiaru = id_p);

idpunktu := (SELECT id_punktu FROM pomiary WHERE idpomiaru = id_p);
```

```
s_o := (SELECT stan_ostrzegawczy FROM punkty_pomiarowe WHERE id_punktu =
idpunktu);

poziomwody = (SELECT poziom_wody FROM pomiary WHERE idpomiaru = id_p);

przekroczenie_stanu = poziomwody - s_o;

INSERT INTO ostrzerzenia(id_ostrzezenia, id_punktu, czas_ostrzezenia,
przekroczony_stan_ostrz, przekroczony_stan_alarm)
VALUES(1963, id_p, czas_ostrz, przekroczenie_stanu, NULL);
END;
$$ LANGUAGE PLpgSQL;
```

25. Napisz funkcje utwory_bez_ocen która wyznacza wszystkie utwory znajdujące się na playlistach klienta które nie otrzymały od niego ocen. Funkcja przyjmuje login klienta i zwraca utwory : idutworu, idalbumu, nazwa, dlugosc

```
CREATE OR REPLACE FUNCTION utwory_bez_ocen(IN f_login VARCHAR(50))
RETURNS TABLE(
    r_idutworu INTEGER,
    r_idalbumu INTEGER,
    r_nazwa VARCHAR(100),
    r_dlugosc INTEGER
) AS
$$

DECLARE id_k INTEGER;
BEGIN
id_k := (SELECT idklienta FROM klienci WHERE login = f_login);
RETURN QUERY
SELECT DISTINCT u.idutworu, u.idalbumu, u.nazwa, u.dlugosc
FROM utwory u
JOIN zawartosc z USING(idutworu)
LEFT JOIN oceny o USING(idutworu)
-- WHERE NOT EXISTS(SELECT 1 FROM oceny oo JOIN utwory uu USING(idutworu)
JOIN zawartosc zz USING(idutworu) WHERE oo.idklienta = id_k AND
zz.idplaylisty IN(SELECT idplaylisty FROM playlisty where idklienta =
id_k))
WHERE idklienta <> id_k
AND z.idplaylisty IN (SELECT idplaylisty FROM playlisty WHERE idklienta =
id_k);
END;
$$ LANGUAGE PLpgSQL;
```

26. Napisz funkcje ocenione_nie_na_playlistach, która wyznacza wszystkie utwory nie znajdujące się na playlistach klienta które otrzymały od niego ocenę. Funkcja przyjmuje id klienta i zwraca nazwy utworów.

```
CREATE OR REPLACE FUNCTION ocenione_nie_na_playlistach(IN id_k INTEGER)
```

```
RETURNS TABLE(
    r_nazwa VARCHAR(100)
) AS
$$
BEGIN
RETURN QUERY
SELECT DISTINCT u.nazwa
FROM utwory u
JOIN oceny o USING(idutworu)
JOIN zawartosc z USING(idutworu)
WHERE o.idklienta = id_k
AND z.idplaylisty NOT IN(SELECT idplaylisty from playlisty where idklienta
= id_k) ;
END;
$$ LANGUAGE PLpgSQL;
```

27. Napisz funkcje dlugie_utwory, która przyjmuje idplaylisty i zwraca łączną liczbę utworów które się na niej znajdują i które trwają powyżej 300 s. Dodatkowo utwory muszą pochodzić z albumów wydanych przed datą urodzenia właściciela playlisty. Jeżeli na playliście nie występuje żaden taki utwór to funkcja zwraca 0;

```
CREATE OR REPLACE FUNCTION dlugie_utwory(IN idp INTEGER)
RETURNS INTEGER AS
$$
DECLARE liczba INTEGER;
BEGIN
liczba := (SELECT COALESCE(COUNT(z.idutworu), 0) FROM zawartosc z JOIN
utwory u USING(idutworu) JOIN albumy a USING(idalbumu) WHERE z.idplaylisty
= idp AND u.dlugosc > 300 AND a.data_wydania < (SELECT data_urodzenia FROM
klienci JOIN playlisty USING(idklienta) WHERE idplaylisty = idp));
RETURN liczba;
END;
$$ LANGUAGE PLpgSQL;
```

28. Napisz funkcję dlugosc_min która zwróci idplaylisty i długość jej trwania w minutach (jedna minuta =60s) zaokrąglij wynik stosując FLOOR(jakaś_liczba), jeśli na playliście nie występują żadne utwory należy ją pominąć. Funkcja przyjmuje nazwę playlisty;

```
    CREATE OR REPLACE FUNCTION dlugosc_min(IN p_n VARCHAR(30))
    RETURNS TABLE(
        r_idplaylisty INTEGER,
        r_dlugosc INTEGER
    ) AS
    $$
    DECLARE idp INTEGER;
    BEGIN
    idp := (SELECT idplaylisty FROM playlisty WHERE nazwa = p_n);
```

```
    RETURN QUERY
    SELECT z.idplaylisty, FLOOR(SUM(u.dlugosc)/60)::INTEGER
    FROM zawartosc z
    JOIN utwory u USING(idutworu)
    WHERE z.idplaylisty = idp
    GROUP BY z.idplaylisty;
    END;
    $$ LANGUAGE PLpgSQL;
```

29. Napisz funkcje, która doda add_new_client do tabeli klienci nowego klienta. Funkcja nie przyjmuje argumentów. Login klienta stanowi login klienta o najwyższym id z dopiskiem "ALT", id jest o jeden większe od obecnego maksymalnego. Nowy klient ma mieć tą samą datę urodzenia co klient który ocenił najwięcej utworów (jeśli takich klientów jest więcej to wybierz datę urodzenia tego z nich który ma najwyższe id), data rejestracji ma być dzisiejszą datą. Funkcja zwraca informacje na temat nowego klienta - idklienta, login, data_rejestracji, data_urodzenia

```
CREATE OR REPLACE FUNCTION add_new_client()
RETURNS TABLE(
    r_idklienta INTEGER,
    r_login VARCHAR(50),
    r_data_rejestracji DATE,
    r_data_urodzenia DATE
)AS
$$
DECLARE login_k VARCHAR(50);
DECLARE data_ur DATE;
DECLARE id_max_ocen INTEGER;
DECLARE id_k INTEGER;
DECLARE r_date DATE;
BEGIN
id_max_ocen := (SELECT COALESCE(COUNT(o.idutworu), 0)::INT FROM klienci k
LEFT JOIN oceny o USING(idklienta)
GROUP BY k.idklienta
ORDER BY 1 DESC, k.idklienta
LIMIT 1
);
data_ur := (SELECT data_urodzenia FROM klienci WHERE idklienta =
id_max_ocen);
login_k := (SELECT login || 'ALT' FROM klienci k WHERE idklienta = (SELECT
MAX(idklienta) FROM klienci));
id_k := (SELECT idklienta +1 FROM klienci k WHERE idklienta = (SELECT
MAX(idklienta) FROM klienci));
r_date := current_date;
INSERT INTO klienci VALUES(id_k, login_k, r_date, data_ur);
RETURN QUERY
SELECT idklienta, login, data_rejestracji, data_urodzenia
FROM klienci
```

```
WHERE idklienta = id_k;
END;
$$ LANGUAGE PLpgSQL;
```

30. Napisz Funkcje polubienia_wykonawcow, która wyświetli id i nazwy wykonawców oraz dla każdego z nich pokaże liczbę polubień (INT) (polubienie: lubi equals true)które zdobyły utwory danego wykonawcy. Funkcja nie przyjmuje argumentów.

```
CREATE OR REPLACE FUNCTION polubienia_wykonawcow()
RETURNS TABLE(
    r_idwykonawcy INTEGER,
    r_nazwa VARCHAR(100),
    r_liczba_pol INTEGER
) AS
$$
BEGIN
RETURN QUERY
SELECT w.idwykonawcy, w.nazwa, COALESCE(SUM(o.lubi::int), 0)::INT
FROM wykonawcy w
LEFT JOIN albumy a USING(idwykonawcy)
LEFT JOIN utwory u USING(idalbumu)
LEFT JOIN oceny o USING(idutworu)
GROUP BY w.idwykonawcy, w.nazwa;
END;
$$ LANGUAGE PLpgSQL;
```