Laboratorium 12

Wyzwalacze, transakcje, zarządzanie uprawnieniami, widoki.

Uwaga: Poniższy zestaw zadań przeznaczony jest na jeden półtoragodzinny blok zajęć oraz pracę samodzielną poza zajęciami laboratoryjnymi. Zadania lub punkty oznaczone symbolem $\star$ przeznaczone są do realizacji samodzielnej. Podczas zajęć laboratoryjnych należy w pierwszej kolejności realizować pozostałe obowiązkowe podpunkty.

Zadanie 12.1

1. Utwórz (i przetestuj działanie) wyzwalacz (w schemacie kwiaciarnia), który przy złożeniu zamówienia przez klienta:

   - ☐ oblicza rabat dla sprzedającego (użyj funkcji z zadania 11.7) i modyfikuje pole cena w dodawanym rekordzie,
   - ☐ zmniejsza liczbę dostępnych kompozycji w tabeli kompozycje,
   - ☐ dodaje rekord do tabeli zapotrzebowanie, jeśli stan danej kompozycji spada poniżej stanu minimalnego.

```
DROP TRIGGER IF EXISTS tr_poZamowieniu ON kwiaciarnia.zamowienia;

CREATE OR REPLACE FUNCTION kwiaciarnia.fn_poZamowieniu()
RETURNS TRIGGER AS
$$
DECLARE rabat INTEGER;
DECLARE stanRoznica INTEGER;
BEGIN

        rabat := kwiaciarnia.rabat(new.idklienta);

        IF rabat > 0 THEN
                new.cena := new.cena - (new.cena *
                                        (rabat::decimal / 100::decimal));
        END IF;

        UPDATE kwiaciarnia.kompozycje
        SET stan = stan - 1
        WHERE idkompozycji = new.idkompozycji;

        SELECT (stan - minimum) INTO stanRoznica
        FROM kwiaciarnia.kompozycje
        WHERE idkompozycji = new.idkompozycji;

        IF (stanRoznica < 0) THEN
                INSERT INTO kwiaciarnia.zapotrzebowanie
                (idkompozycji, data)
                VALUES
                (new.idkompozycji, CURRENT_DATE)
                ON CONFLICT (idkompozycji) DO UPDATE SET data = CURRENT_DATE;
        END IF;

        RETURN new;
```

```
END;
$$ LANGUAGE PLpgSQL;

CREATE TRIGGER tr_poZamowieniu
BEFORE INSERT ON kwiaciarnia.zamowienia
FOR EACH ROW
EXECUTE PROCEDURE kwiaciarnia.fn_poZamowieniu();
```

2. ⋆ Utwórz wyzwalacz (w schemacie kwiaciarnia), który automatycznie usuwa rekordy z tabeli zapotrzebowanie, jeżeli po dostawie (after update) wzrasta stan danej kompozycji powyżej minimum. Przetestuj działanie wyzwalacza.

```
DROP TRIGGER IF EXISTS tr_anuluj_zapotrzebowanie ON kwiaciarnia.kompozycje;
CREATE OR REPLACE FUNCTION kwiaciarnia.fn_anuluj_zapotrzebowanie()
RETURNS TRIGGER AS
$$

BEGIN
        IF old.stan < new.minimum AND new.stan > new.minimum THEN
                DELETE FROM zapotrzebowanie
                        WHERE idkompozycji = new.idkompozycji;
        END IF;

        RETURN new;
END;

$$
LANGUAGE plpgsql;

CREATE TRIGGER tr_anuluj_zapotrzebowanie
AFTER UPDATE ON kwiaciarnia.kompozycje
FOR EACH ROW
EXECUTE PROCEDURE kwiaciarnia.fn_anuluj_zapotrzebowanie();
```

Zadanie 12.2

1. Utwórz wyzwalacz modyfikujący (po aktualizacji rekordów w tabeli pudelka) pole cena w tabeli pudelka, jeżeli cena jest mniejsza niż 105% kosztów wytworzenia danego pudełka czekoladek (koszt wytworzenia czekoladek + koszt pudełka 0,90 zł). W takim przypadku cenę należy ustawić na kwotę 105% kosztów wytworzenia.

```
DROP TRIGGER IF EXISTS tr_poAktualizacji ON public.pudelka;

CREATE OR REPLACE FUNCTION public.fn_poAktualizacji()
RETURNS TRIGGER AS
$$
DECLARE koszt_wytw NUMERIC(7,2);
BEGIN

        SELECT SUM(cz.koszt * z.sztuk) + 0.9 INTO koszt_wytw
        FROM
                pudelka p
                INNER JOIN zawartosc z ON p.idpudelka = z.idpudelka
                INNER JOIN czekoladki cz ON cz.idczekoladki = z.idczekoladki
        WHERE
```

```
                p.idpudelka = new.idpudelka
        GROUP BY p.idpudelka;

        IF new.cena < 1.05*koszt_wytw THEN
                new.cena = 1.05*koszt_wytw;
        END IF;

        RETURN new;

END;
$$ LANGUAGE PLpgSQL;

CREATE TRIGGER tr_poAktualizacji
BEFORE UPDATE ON pudelka
FOR EACH ROW
EXECUTE PROCEDURE fn_poAktualizacji();
```

2. ⋆ Utwórz wyzwalacz modyfikujący (przy wstawianiu i aktualizacji rekordów w tabeli zawartosc) pole cena w tabeli pudelka, jeżeli cena jest mniejsza niż 105% kosztów wytworzenia danego pudełka czekoladek (koszt wytworzenia czekoladek + koszt pudełka 0,90 zł). W takim przypadku cenę należy ustawić na kwotę 105% kosztów wytworzenia.

```
DROP TRIGGER IF EXISTS tr_cena_pudelka1 ON pudelka;

CREATE OR REPLACE FUNCTION fn_cena_pudelka1()
RETURNS TRIGGER AS
$$

DECLARE
        koszt_wytw NUMERIC(7,2);
        stara_cena NUMERIC(7,2);
        nowa_cena NUMERIC(7,2);

BEGIN
        SELECT SUM(koszt*sztuk)+0.9 INTO koszt_wytw
                FROM pudelka p
                INNER JOIN zawartosc z USING(idpudelka)
                INNER JOIN czekoladki c USING(idczekoladki)
        WHERE idpudelka = new.idpudelka
        GROUP BY cena;

        SELECT cena INTO stara_cena
        FROM pudelka
                WHERE idpudelka = new.idpudelka;

        IF stara_cena < 1.05*koszt_wytw THEN
                nowa_cena := 1.05*koszt_wytw;
        ELSE
                nowa_cena := stara_cena;
        END IF;

        UPDATE pudelka SET cena=nowa_cena
                WHERE idpudelka = new.idpudelka;

        RETURN new;
```

```
END;

$$
LANGUAGE plpgsql;

CREATE TRIGGER tr_cena_pudelka1
BEFORE INSERT OR UPDATE ON zawartosc
FOR EACH ROW EXECUTE PROCEDURE fn_cena_pudelka1();
```

3. ⋆ Utwórz wyzwalacz modyfikujący (przy aktualizacji rekordów w tabeli czekoladki) pole cena w tabeli pudelka, jeżeli cena jest mniejsza niż 105% kosztów wytworzenia danego pudełka czekoladek (koszt wytworzenia czekoladek + koszt pudełka 0,90 zł). W takim przypadku cenę należy ustawić na kwotę 105% kosztów wytworzenia.

```
DROP TRIGGER IF EXISTS tr_cena_pudelka2 ON pudelka;

CREATE OR REPLACE FUNCTION fn_cena_pudelka2()
RETURNS TRIGGER AS
$$

DECLARE
wiersz RECORD;
koszt1 NUMERIC(7,2);
cena1 NUMERIC(7,2);

BEGIN
        FOR wiersz IN
                SELECT idpudelka FROM zawartosc z
                        WHERE z.idczekoladki = new.idczekoladki
        LOOP
                SELECT SUM(koszt*sztuk) + 0.9 INTO koszt1
                        FROM pudelka p
                        INNER JOIN zawartosc z USING(idpudelka)
                        INNER JOIN czekoladki c USING(idczekoladki)
                WHERE idpudelka = wiersz.idpudelka
                GROUP BY cena;

                SELECT cena INTO cena1
                FROM pudelka
                WHERE idpudelka = wiersz.idpudelka;

                IF cena1 < 1.05*koszt1 THEN
                        cena1 := 1.05*koszt1;
                END IF;

                UPDATE pudelka SET cena=cena1
                WHERE idpudelka = wiersz.idpudelka;

        END LOOP;

        RETURN new;
END;

$$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER tr_cena_pudelka2
AFTER UPDATE ON czekoladki
FOR EACH ROW EXECUTE PROCEDURE fn_cena_pudelka2();
```

Zadanie 12.3

1. Rozpocznij transakcję: begin.
2. Wykonaj szereg dowolnych zapytań DML (insert/update/delete).
3. Sprawdź czy efekty w/w zapytań są widoczne w bazie danych (w tej samej sesji oraz w nowo utworzonej sesji).
4. Anuluj transakcję: rollback.
5. Sprawdź czy efekty zapytań z punktu 3 są wciąż widoczne w bazie danych (w tej samej sesji oraz w nowo utworzonej sesji).
6. Powtórz kroki 1-3. Zatwierdź transakcję: commit.
7. Sprawdź czy efekty zapytań z punktu 6 są wciąż widoczne w bazie danych (w tej samej sesji oraz w nowo utworzonej sesji).
8. $\star$ wykonaj powyższe testy zmieniając poziom izolacji transakcji http://www.postgresql.org/docs/9.1/static/sql-set-transaction.html.

Zadanie 12.4

Udzielenie dostępu:

```
GRANT prawa ON obiekt TO uzytkownik;
```

☐ prawa: select, insert, update, delete, create, all privileges,

☐ obiekt: tabela, schemat, baza danych, widok etc.,

☐ uzytkownik: nazwa uzytkownika, public (wszyscy).

Odbieranie praw:

```
REVOKE prawa ON obiekt FROM uzytkownik;
```

Informacje o prawach dostępu z poziomu psql można uzyskać za pomocą polecenia \z (albo \dp). Domyślnie prawa dostępu są tylko dla właściciela!!!

1. Udziel dostępu do wykonywania zapytań select 2-giej osobie w grupie do tabeli kompozycje (schemat kwiaciarnia); przetestuj.

   ```
   CREATE ROLE nazwa;

   GRANT SELECT ON kwiaciarnia.kompozycje TO nazwa;
   ```

2. Udziel dostępu do wykonywania zapytań select 2-giej osobie w grupie do schematu firma; przetestuj.

```
CREATE SCHEMA firma;
GRANT SELECT ON firma.dzialy TO nazwa;
GRANT SELECT ON firma.pracownicy TO nazwa;
```

3. Udziel dostępu do dodawania, usuwania, modyfikacji wierszy 2-giej osobie w grupie do tabeli klienci (schemat kwiaciarnia); przetestuj.

```
GRANT INSERT, DELETE, UPDATE ON kwiaciarnia.klienci TO nazwa;
```

4. Zezwól 2-giej osobie na tworzenie nowych obiektów w schemacie firma; przetestuj.

```
GRANT CREATE ON SCHEMA firma TO nazwa;
```

5. Powtórz ćwiczenie zamieniając się rolami.

6. Usuń przyznane prawa. Sprawdź, czy 2-ga osoba może teraz wykonać jakieś operacje.

```
REVOKE SELECT ON kwiaciarnia.kompozycje FROM nazwa;
```

Zadanie 12.5

1. Utwórz widok (perspektywę), który zwraca: identyfikator zamówienia, datę realizacji, nazwę i adres klienta dla każdego zamówienia - zapytanie takie może być używane np. przez dział wysyłki.

```
CREATE OR REPLACE VIEW zamowieniaWysylka AS
SELECT
        z.idzamowienia, z.datarealizacji,
        k.nazwa, k.ulica, k.miejscowosc, k.kod
FROM
        zamowienia z
INNER JOIN klienci k USING(idklienta);
```

2. Udostępnij widok z poprzedniego zadania innemu użytkownikowi do odczytu (pracownikowi działu zamówień ;))

```
GRANT SELECT ON wysylki TO nazwa;
```

Zadanie 12.6

1. Wykonaj kopie zapasową wszystkich schematów Twojej bazy danych wykorzystując polecenie pg_dump. Przećwicz opcje: -a oraz –inserts. Zobacz co jest rezultatem działania w/w polecenia. Jak odtworzyć taką kopię?

2. ⋆ Wykonaj kopie danych Internetowej Kwiaciarni (uwaga: tylko dane i obiekty z jej schematu).

3. ⋆ Odtwórz kwiaciarnię w innym schemacie.