author: Kacper Bohaczyk
version: 21-02-2024

Datenbankseitige Programmierung

Aufgabe 1

```
CREATE OR REPLACE FUNCTION bilanz(client_id INTEGER)
RETURNS INTEGER
AS $$
DECLARE
einzahlungen DECIMAL;
auszahlungen DECIMAL;
BEGIN
SELECT SUM(amount) INTO einzahlungen FROM transfers WHERE to_client_id
= client_id;
SELECT SUM(amount) INTO auszahlungen FROM transfers WHERE
from_client_id = client_id;
RETURN einzahlungen - auszahlungen;
END;
$$ LANGUAGE plpgsql;
```

In folgendem Dokument wird die Funktion "bilianz" erweitert in Form von bilanz_mit_steuer

Aufgabe 1: Erweitere die Funktion 'bilanz' wie folgt: Die Bank hat bis 2019 bei jeder Transaktion eine Steuer von 2% eingehoben - d.h. ein Transfer von 100.- soll in der Bilanz des Empfaengers nur mit 98.- gewertet werden. 2020 wurde diese Steuer auf 1% gesenkt. Erstelle eine Funktion bilanz_mit_steuer, die diese beiden Steuersaetze beruecksichtigt. (Hint: mit `date_part('year',date)` laesst sich das Jahr zu einem Datum ermitteln.) Rufe deine Funktion mit `select bilanz(<id>)` auf und teste anhand von passenden Daten, ob sie auch funktioniert.

```
CREATE OR REPLACE FUNCTION bilanz_mit_steuer(client_id INTEGER)
RETURNS DECIMAL
AS $$
DECLARE
    einzahlungen DECIMAL;
    auszahlungen DECIMAL;
    steuersatz DECIMAL;
BEGIN
    IF date_part('year', current_date) < 2020 THEN
        steuersatz := 0.02; -- Steuersatz bis 2019
    ELSE
        steuersatz := 0.01; -- Steuersatz ab 2020
    END IF;

    SELECT COALESCE(SUM(amount), 0) INTO einzahlungen FROM transfers WHERE
to_client_id = client_id;
    SELECT COALESCE(SUM(amount), 0) INTO auszahlungen FROM transfers WHERE
from_client_id = client_id;

    RETURN (einzahlungen - auszahlungen) * (1 - steuersatz);
END;
```

```
$$ LANGUAGE plpgsql;
```

Aufgabe 2

In dieser Aufgabe wird Geld von einem Sender zu einem Empfänger übertragen. Error bei größeren Betrag als vorhanden

Aufgabe 2: Passe die Procedure transfer_direct() so an, dass die Ueberweisung nicht durchgefuehrt wird, falls das Konto nicht gedeckt ist, d.h., wenn am Konto des Senders weniger Geld vorhanden ist, als ueberwiesen werden soll. (Hint 1: mit zum Beispiel `IF a < b THEN ... END IF;` lassen sich Fallunterscheidungen durchfuehren.) (Hint 2: mit `RAISE EXCEPTION 'meine Fehlermeldung'` laesst sich die Procedure abbrechen.)

```
-- Drop the procedure if it already exists
DROP PROCEDURE IF EXISTS transfer_direct(sender INTEGER, recipient INTEGER,
howmuch DECIMAL);

-- Create or replace the procedure for direct transfer
CREATE OR REPLACE PROCEDURE transfer_direct(sender INTEGER, recipient INTEGER,
howmuch DECIMAL)
AS $$
BEGIN
DECLARE
    sender_balance DECIMAL;
BEGIN
    SELECT amount INTO sender_balance FROM accounts WHERE client_id = sender;
    IF sender_balance >= howmuch THEN
        UPDATE accounts SET amount = amount - howmuch WHERE client_id = sender;
        UPDATE accounts SET amount = amount + howmuch WHERE client_id =
recipient;
        COMMIT;
    ELSE
        RAISE EXCEPTION 'Konto des Senders nicht gedeckt für die Überweisung von
%.2f', howmuch;
    END IF;
END;
END;
$$ LANGUAGE plpgsql;
```

Aufgabe 3

Es wird eine Trigger-Funktion erstellt die automatisch bei einem Transaktions-Insert die Kontostände der Clients ändert

Aufgabe 3: Erstelle einen Trigger, welcher bewirkt, dass bei dem Anlegen eines neuen Transfers der aktuelle Kontostand in der accounts-Tabelle automatisch angepasst wird. Erstelle dafuer eine Function update_accounts(), die die noetigen Anpassungen durchfuehrt und mache diese Funktion mittels `CREATE TRIGGER new_transfer BEFORE INSERT ON transfers FOR EACH ROW EXECUTE PROCEDURE update_accounts();` zum Trigger. (Hint: Damit der Transfer auch gespeichert wird, muss dein Trigger RETURN NEW zurueckgeben);

```
-- Drop the function if it already exists
```

```
DROP FUNCTION IF EXISTS update_accounts() CASCADE;

-- Create or replace the function for updating accounts
CREATE OR REPLACE FUNCTION update_accounts()
RETURNS trigger
AS $$
BEGIN
    UPDATE accounts SET amount = amount - NEW.amount WHERE client_id =
NEW.from_client_id;
    UPDATE accounts SET amount = amount + NEW.amount WHERE client_id =
NEW.to_client_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Aufgabe 4

Dies ist eine Erweiterung von Aufgabe 3. Diesmal wird eine IF-Abfrage verwendet um zu testen ob die Transaktion nicht über den Kontostand des Senders geht.

Aufgabe 4: Passe den Trigger aus Aufgabe 3 so an, dass nur Transfers angenommen werden, die das Senderkonto nicht ueberziehen. D.h. ueberpruefe, ob das Konto nach der Ueberweisung unter 0.- fallen wuerde und verhindere in diesem Fall mit RETURN NULL das Einfuegen der Transaktion

```
-- Drop the function if it already exists
DROP FUNCTION IF EXISTS update_accounts() CASCADE;

-- Create or replace the function for updating accounts
CREATE OR REPLACE FUNCTION update_accounts()
RETURNS trigger
AS $$
DECLARE
    kontostand DECIMAL;
BEGIN
    SELECT amount INTO kontostand FROM accounts WHERE client_id =
NEW.from_client_id;
    IF NEW.amount > kontostand THEN
        RETURN NULL;
    END IF;
    UPDATE accounts SET amount = amount - NEW.amount WHERE client_id =
NEW.from_client_id;
    UPDATE accounts SET amount = amount + NEW.amount WHERE client_id =
NEW.to_client_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
-- Drop the trigger if it already exists
DROP TRIGGER IF EXISTS new_transfer ON transfers;

CREATE TRIGGER new_transfer BEFORE INSERT ON transfers FOR EACH ROW EXECUTE
PROCEDURE update_accounts();
```

EK

```
-- Erstellen der Tabelle 'statistics'
CREATE TABLE IF NOT EXISTS statistics (
    date date PRIMARY KEY,
    sum_amount decimal,
    avg_amount decimal,
    max_amount decimal,
    taxes decimal
);

-- Erstellen der Prozedur 'calc_statistics'
CREATE OR REPLACE PROCEDURE calc_statistics()
AS $$
DECLARE
    cur_date DATE;
    total_sum DECIMAL;
    avg_amount DECIMAL;
    max_amount DECIMAL;
    total_taxes DECIMAL;
    transfer_dates CURSOR FOR SELECT DISTINCT transfer_date FROM transfers;
BEGIN
    -- Leeren der Tabelle 'statistics' vor der Berechnung
    TRUNCATE TABLE statistics;

    -- Öffnen des Cursors
    OPEN transfer_dates;

    LOOP
        -- Abrufen des nächsten Datums
        FETCH transfer_dates INTO cur_date;
        EXIT WHEN NOT FOUND;

        -- Berechnen der Gesamtsumme der Überweisungen für den aktuellen Tag
        SELECT SUM(amount) INTO total_sum
        FROM transfers
        WHERE transfer_date = cur_date;

        -- Berechnen des Durchschnittsbetrags der Überweisungen für den aktuellen
Tag
        SELECT AVG(amount) INTO avg_amount
        FROM transfers
        WHERE transfer_date = cur_date;

        -- Berechnen der höchsten Überweisung für den aktuellen Tag
        SELECT MAX(amount) INTO max_amount
        FROM transfers
```

```
        WHERE transfer_date = cur_date;

        -- Berechnen der Steuern für den aktuellen Tag
        SELECT CASE
            WHEN EXTRACT(YEAR FROM cur_date) <= 2019 THEN
                SUM(amount * 0.02)
            ELSE
                SUM(amount * 0.01)
        END INTO total_taxes
        FROM transfers
        WHERE transfer_date = cur_date;

        -- Einfügen der berechneten Werte in die Tabelle 'statistics'
        INSERT INTO statistics (date, sum_amount, avg_amount, max_amount, taxes)
        VALUES (cur_date, total_sum, avg_amount, max_amount, total_taxes);

        RAISE NOTICE 'Statistics calculated and inserted for date: %', cur_date;
    END LOOP;

    -- Schließen des Cursors
    CLOSE transfer_dates;
END;
$$ LANGUAGE plpgsql;

-- Testdaten einfügen
-- Löschen der Tabellen, falls vorhanden
DROP TABLE IF EXISTS transfers;
DROP TABLE IF EXISTS accounts;

-- Erstellen der Tabelle für Transfers
CREATE TABLE transfers (
    transfer_id SERIAL PRIMARY KEY,
    from_client_id INTEGER NOT NULL,
    to_client_id INTEGER NOT NULL,
    transfer_date DATE NOT NULL,
    amount DECIMAL NOT NULL
);

-- Beispiel-Daten für Transfers
INSERT INTO transfers (from_client_id, to_client_id, transfer_date, amount)
VALUES
(1, 2, '2023-05-20', 200.00),
(2, 1, '2023-05-20', 100.00),
(3, 1, '2023-05-21', 500.00),
(1, 3, '2023-05-21', 300.00),
(1, 2, '2019-12-31', 150.00),
(2, 1, '2020-01-01', 250.00);

-- Ausführen der Prozedur, um die Statistik-Daten zu berechnen und einzufügen
CALL calc_statistics();

-- Überprüfen der eingefügten Daten
SELECT * FROM statistics;

SELECT * FROM transfers;
```

Aufgabe 1

```sql
SELECT bilanz(1);
SELECT bilanz_mit_steuer(1);

```

Aufgabe 2

```sql
CALL transfer_direct(1, 2, 200.00);

```

Aufgabe 3

```sql
-- Drop the trigger if it already exists
DROP TRIGGER IF EXISTS new_transfer ON transfers;

-- Erstelle den Trigger für Aufgabe 3
CREATE TRIGGER new_transfer BEFORE INSERT ON transfers FOR EACH ROW EXECUTE
PROCEDURE update_accounts();

```

Query | Query History

```
90 (2, 1, '2020-01-01', 250.00);
91
92 -- Ausführen der Prozedur, um die Statistik-Daten zu berechnen und ein
93 CALL calc_statistics();
94
95 -- Überprüfen der eingefügten Daten
96 SELECT * FROM statistics;
97
98 SELECT * FROM transfers;
```

Data Output | Messages | Notifications

	transfer_id [PK] integer	from_client_id integer	to_client_id integer	transfer_date date	amount numeric
1	1	1	2	2023-05-20	200.00
2	2	2	1	2023-05-20	100.00
3	3	3	1	2023-05-21	500.00
4	4	1	3	2023-05-21	300.00
5	5	1	2	2019-12-31	150.00
6	6	2	1	2020-01-01	250.00

Query | Query History

```
89 (1, 2, '2019-12-31', 150.00),
90 (2, 1, '2020-01-01', 250.00);
91
92 -- Ausführen der Prozedur, um die Statistik-Daten zu berechnen und ein
93 CALL calc_statistics();
94
95 -- Überprüfen der eingefügten Daten
96 SELECT * FROM statistics;
97
```

Data Output | Messages | Notifications

	date date	sum_amount numeric	avg_amount numeric	max_amount numeric	taxes numeric
1	2023-05-21	800.00	400.0000000000000000	500.00	8.0000
2	2023-05-20	300.00	150.0000000000000000	200.00	3.0000
3	2020-01-01	250.00	250.0000000000000000	250.00	2.5000
4	2019-12-31	150.00	150.0000000000000000	150.00	3.0000