Trigger

Christof Zlabinger

GK

Aufgabe 1

```
 CREATE OR REPLACE FUNCTION bilanz_mit_steuer(client_id INTEGER)
RETURNS DECIMAL
AS $$
DECLARE
   einzahlungen DECIMAL;
   auszahlungen DECIMAL;
   steuer_satz DECIMAL;
BEGIN
   SELECT SUM(CASE WHEN date_part('year', date) <= 2019 THEN amount * 0.98 ELSE amount * 0.99 END) INTO einzahlungen FROM transfers
   SELECT SUM(CASE WHEN date_part('year', date) <= 2019 THEN amount * 0.98 ELSE amount * 0.99 END) INTO auszahlungen FROM transfers
   RETURN einzahlungen - auszahlungen;
END;
$$ LANGUAGE plpgsql;
```

Aufgabe 2

```
 CREATE OR REPLACE PROCEDURE transfer_direct(sender INTEGER, recipient INTEGER, howmuch DECIMAL)
AS $$
DECLARE
    sender_balance DECIMAL;
BEGIN
    -- Überprüfen, ob der Sender genügend Geld hat
    SELECT amount INTO sender_balance FROM accounts WHERE client_id = sender;
    IF sender_balance < howmuch THEN
        RAISE EXCEPTION 'Nicht genügend Geld auf dem Konto des Senders';
    ELSE
        -- Wenn genügend Geld vorhanden ist, führen Sie die Überweisung durch
        UPDATE accounts SET amount = amount - howmuch WHERE client_id = sender;
        UPDATE accounts SET amount = amount + howmuch WHERE client_id = recipient;
        COMMIT;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Aufhabe 3

```
CREATE OR REPLACE FUNCTION update_accounts()
RETURNS TRIGGER AS $$
BEGIN
    -- Aktualisieren des Kontostands des Senders
    UPDATE accounts
    SET amount = amount - NEW.amount
    WHERE client_id = NEW.from_client_id;

    -- Aktualisieren des Kontostands des Empfängers
    UPDATE accounts
    SET amount = amount + NEW.amount
    WHERE client_id = NEW.to_client_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


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

Aufgabe 4

```
CREATE OR REPLACE FUNCTION update_accounts()
RETURNS TRIGGER AS $$
DECLARE
    sender_balance DECIMAL;
BEGIN
    SELECT amount INTO sender_balance FROM accounts WHERE client_id = NEW.from_client_id;
    IF sender_balance < NEW.amount THEN
        RAISE EXCEPTION 'Nicht genügend Geld auf dem Konto des Senders';
    ELSE
        -- Aktualisieren des Kontostands des Senders
        UPDATE accounts
        SET amount = amount - NEW.amount
        WHERE client_id = NEW.from_client_id;

        -- Aktualisieren des Kontostands des Empfängers
        UPDATE accounts
        SET amount = amount + NEW.amount
        WHERE client_id = NEW.to_client_id;

        RETURN NEW;
    END IF;
END;
$$ LANGUAGE plpgsql;


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

EK

```
use postgres::{Client, NoTls, Error};
use tabled::{Tabled, Table};
use chrono::NaiveDate;
use rust_decimal::Decimal;

fn main() -> Result<(), Error> {
```

```
fn main() -> Result<(), Error> {
    let mut client = Client::connect("postgresql://postgres:postgres@localhost/kontodaten", NoTls)?;

    client.batch_execute("
        CREATE OR REPLACE FUNCTION calc_statistics() RETURNS VOID AS $$
        DECLARE
            current_date DATE;
            sum_amount DECIMAL;
            avg_amount DECIMAL;
            max_amount DECIMAL;
            taxes DECIMAL;
        BEGIN
            FOR current_date IN SELECT DISTINCT date FROM transfers WHERE date = current_date LOOP
                RAISE NOTICE 'Processing date: %', current_date;

                SELECT SUM(amount) INTO sum_amount FROM transfers WHERE date = current_date;
                SELECT AVG(amount) INTO avg_amount FROM transfers WHERE date = current_date;
                SELECT MAX(amount) INTO max_amount FROM transfers WHERE date = current_date;
                SELECT SUM(amount * CASE WHEN date < '2020-01-01' THEN 0.02 ELSE 0.01 END) INTO taxes FROM transfers WHERE date = cu

                RAISE NOTICE 'Sum: %, Avg: %, Max: %, Taxes: %', sum_amount, avg_amount, max_amount, taxes; -- Debugging-Information

                INSERT INTO statistics (date, sum_amount, avg_amount, max_amount, taxes)
                VALUES (current_date, sum_amount, avg_amount, max_amount, taxes);
            END LOOP;
        END;
        $$ LANGUAGE plpgsql;
    ")?;

    client.batch_execute("SELECT calc_statistics();")?;

    let rows = client.query("SELECT * FROM statistics", &[]).unwrap();

    // Convert the rows into a vector of your struct
    let data: Vec<Statistics> = rows.iter().map(|row| {
        Statistics {
            date: row.get("date"),
            sum_amount: row.get("sum_amount"),
            avg_amount: row.get("avg_amount"),
            max_amount: row.get("max_amount"),
            taxes: row.get("taxes"),
        }
    }).collect();

    // Print the data as a table
    let table = Table::new(&data);
    println!("{}", table);

    Ok(())
}

#[derive(Tabled)]
pub struct Statistics {
    date: NaiveDate,
    sum_amount: Decimal,
    avg_amount: Decimal,
    max_amount: Decimal,
    taxes: Decimal,
}
```

```
[package]
name = "triggeer"
version = "0.1.0"
edition = "2021"

[dependencies]
postgres = { version = "0.19.7", features = ["with-chrono-0_4"] }
tabled = "0.15.0"
chrono = "0.4.38"
rust_decimal = { version = "1.35.0", features = ["db-postgres"] }
```