Datenbankseitige Programmierung

Markus Rafeiner, Leonhard Stransky, Julian Neuwirth, Marvin Kasteiner, Timur Visne, 26.02.2024

Serverseitige Datenbankprogrammierung in PostgreSQL GK

Man kann auch Code auf der Datenbank selbst ausführen. Das ist in der Praxis schneller, als alles am Client rechnen zu lassen. Es spart auch an Datenübertragung, da man einfach das Ergebnis überliefern kann, anstatt alle dafür notwendigen Daten.

Es gibt 3 verschiedene Möglichkeiten: Functions, Procedures und Trigger.

Funktionen werden mit `CREATE OR REPLACE FUNCTION` erstellt und dann auf der Datenbank gespeichert.

Procedures sind sehr ähnlich, aber sie können eigene Transactions kontrollieren, anstatt dass alles in einer Transaction abläuft.

Triggers sind eine spezielle Art von Funktionen, welche vom DBMS automatisch aufgerufen werden, wenn bestimmte Ereignisse eintreten

Aufgabe 1

> **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;
BEGIN
    -- Einzahlungen mit Steuer
    SELECT SUM(CASE
                WHEN date_part('year', date) <= 2019 THEN amount * 0.98
                ELSE amount * 0.99
            END) INTO einzahlungen
    FROM transfers
    WHERE to_client_id = client_id;

    -- Auszahlungen ohne Steuer
    SELECT SUM(amount) INTO auszahlungen
    FROM transfers
    WHERE from_client_id = client_id;

    -- Berechnung der Bilanz
    RETURN einzahlungen - auszahlungen;
END;
$$ LANGUAGE plpgsql;
```

Aufgabe 2

> **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.)

```
CREATE OR REPLACE PROCEDURE transfer_direct(sender INTEGER, recipient INTEGER, howmuch
DECIMAL)
AS $$
DECLARE
    sender_balance DECIMAL;
BEGIN
    -- Ermittlung des aktuellen Guthabens des Senders
    SELECT amount INTO sender_balance FROM accounts WHERE client_id = sender;

    -- Überprüfung, ob das Guthaben für die Überweisung ausreicht
    IF sender_balance < howmuch THEN
        -- Auslösen einer Exception, wenn das Guthaben nicht ausreicht
        RAISE EXCEPTION 'Nicht genügend Guthaben für die Überweisung. Verfügbar: %,
erforderlich: %', sender_balance, howmuch;
    ELSE
        -- Durchführung der Überweisung, wenn das Guthaben ausreicht
        UPDATE accounts SET amount = amount - howmuch WHERE client_id = sender;
        UPDATE accounts SET amount = amount + howmuch WHERE client_id = recipient;
    END IF;
    COMMIT;
END;
$$ LANGUAGE plpgsql;
```

Aufgabe 3

> **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);

```
CREATE OR REPLACE FUNCTION update_accounts()
RETURNS TRIGGER AS $$
BEGIN
    -- Erhöhung des Kontostands des Empfängers
    UPDATE accounts SET amount = amount + NEW.amount WHERE client_id =
NEW.to_client_id;

    -- Verringerung des Kontostands des Absenders
    UPDATE accounts SET amount = amount - NEW.amount WHERE client_id =
NEW.from_client_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;



-- Implementing Trigger
CREATE TRIGGER new_transfer BEFORE INSERT ON transfers
FOR EACH ROW EXECUTE FUNCTION update_accounts();
```

Aufgabe 4

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

```
CREATE OR REPLACE FUNCTION update_accounts_safe()
RETURNS TRIGGER AS $$
DECLARE
    sender_balance DECIMAL;
BEGIN
    -- Ermittlung des aktuellen Guthabens des Senders
    SELECT amount INTO sender_balance FROM accounts WHERE client_id =
NEW.from_client_id;

    -- Überprüfung, ob das Guthaben ausreicht
    IF sender_balance < NEW.amount THEN
        -- Wenn das Guthaben nicht ausreicht, wird die Transaktion nicht durchgeführt
        RAISE EXCEPTION 'Nicht genügend Guthaben für die Überweisung. Verfügbar: %,
erforderlich: %', sender_balance, NEW.amount;
    ELSE
        -- Ansonsten Durchführung der Kontostandanpassung
        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;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;



-- Implementing Trigger
DROP TRIGGER IF EXISTS new_transfer ON transfers;

CREATE TRIGGER new_transfer BEFORE INSERT ON transfers
FOR EACH ROW EXECUTE FUNCTION update_accounts_safe();
```