Fundusze Europejskie
Wiedza Edukacja Rozwój

Unia Europejska
Europejski Fundusz Społeczny

Programowanie zespołowe 2.0

Programowanie po stronie serwera bazodanowego

Scenariusz zajęć

Czas realizacji: 180 minut
Cele ogólne: Automatyzacja działania serwera bazodanowego.
Cel szczegółowy: Opracowanie procedur i funkcji usprawniających pracę z bazą danych w przygotowywanym projekcie.
Konieczne wiadomości wstępne: Podstawy programowania w dowolnym języku.
Metoda prowadzenia zajęć: Dyskusja z elementami wykładu.

Serwer bazodanowy, poza odpowiadaniem na zapytania użytkownika, może również automatycznie wykonywać pewne czynności. Może sprawdzać poprawność wprowadzanych danych (jeżeli ograniczenia w tabelach nie są wystarczające), przetwarzać je, archiwizować, itp.

Aby zaprogramować serwer do wykonywania tych czynności, najczęściej wykorzystuje się język PL/SQL – język programowania dla SQL-a (w przypadku bazy PostgreSQL jest to odmiana PL/pgSQL). W PL/SQL-u możemy programować procedury i funkcje, które wywoływane przez użytkownika usprawniają jego pracę z bazą. Szczególnym typem procedur są tzw. wyzwalacze – procedury, które wywoływane są automatycznie przy wykonywaniu pewnych akcji na bazie.

Zanim powiemy więcej o samym programowaniu, uzupełnijmy nasze narzędzia języka SQL o dwie dodatkowe rzeczy ułatwiające pracę z bazą.

Kopiowanie danych i widoki

Archiwizacja danych i tworzenie kopii zapasowych na ogół odbywa się przez skopiowanie informacji z jednej tabeli, to innej. Zapytanie wykonujące tę czynność ma następującą strukturę:

```
INSERT INTO DocelowaTabela
SELECT Atrybuty
FROM ŹródłowaTabela
WHERE ... ;
```

Od słowa kluczowego `SELECT` zaczyna się zapytanie wybierające dane, które chcemy zapisać do `DocelowaTabela`. Należy tutaj pamiętać, że kopiowane dane muszą mieć odpowiednią strukturę i typy danych.

Przykład 1. Poniższe zapytanie kopiuje wypożyczenia starsze niż rok (365 dni) do tabeli `WypozyczeniaArch`

```
INSERT INTO WypozyczeniaArch
SELECT *
FROM Wypozyczenia
WHERE CURRENT_DATE-DataWypozyczenia > 365;
```

Ćwiczenie 1. Jakie dane powinny być archiwizowane lub kopiowane w tworzonym projekcie? Stwórz odpowiednie tabele przechowujące kopie wpisów oraz napisz zapytania kopiujące odpowiednie krotki.

Z różnych względów czasami wygodnie jest przechowywać dane częściowo przetworzone (obliczone pewne wartości z danych surowych, zestawienia wymagające łączenia tabel, itp.). Takie dane można zapisywać w tabelach pomocniczych, jednak przy każdej aktualizacji danych w bazie musielibyśmy aktualizować również tabele pomocnicze. Wygodniejszym sposobem jest zapamiętanie zapytania tworzącego tabelę pomocniczą. Jest to tzw. widok. W momencie wywołania widoku wykonuje się zapamiętane zapytanie i zwracane są aktualne dane.

Widoki tworzymy zapytaniem o następującej strukturze:

```
CREATE VIEW NazwaWidoku(NazwaAtr1, NazwaAtr2, ..., NazwaAtrN)
AS
SELECT Atrybut1, Atrybut2, ..., AtrybutN
FROM ... ;
```

gdzie po słowie `AS` mamy zapytanie zwracające interesujące nas dane. Jeżeli nie podamy nazw atrybutów w widoku, zostaną one odziedziczone z zapytania wybierającego dane. W przypadku przetwarzania danych mogą być one problematyczne, więc dobrym zwyczajem jest podawanie nazw atrybutów po nazwie widoku.

Z widoków korzystamy w taki sam sposób jak ze zwykłych tabel.

Ćwiczenie 2. Wykonaj zapytanie

```
SELECT *
FROM PelneWypozyczenia;
```

Jaki jest jego wynik? Dodaj do bazy nową transakcję i wykonaj je jeszcze raz. Czy wynik nowa transakcja pojawiła się w widoku?

Przykład 2. Dla uzyskania pełnego obrazu transakcji w naszej wypożyczalni filmów musimy połączyć wszystkie 3 tabele. Dlatego wygodnie będzie korzystać z widoku, który zapamiętuje takie złączenie:

```
CREATE VIEW PelneWypozyczenia (Id, Imie, Nazwisko, Tytul, Data, Cena,
                                                          Netto, Podatek)
AS
SELECT Id, Imie, Nazwisko, Tytul, DataWypozyczenia, Cena,
                           ROUND(Cena/1.23,2), Cena - ROUND(Cena/1.23,2)
FROM Wypozyczenia JOIN Uzytkownicy USING(IdUzytkownika)
JOIN Filmy USING(IdFilmu);
```

Wypisanie wszystkich informacji o wypożyczeniach z ostatniego miesiąca wygląda teraz jak poniżej:

```
SELECT Id, Imie, Nazwisko, Tytul, Data, Cena
FROM PelneWypozyczenia
WHERE Data + 30 >= CURRENT_DATE;
```

W formie widoków przechowuje się również różnego rodzaju zestawienia. Poniżej dwa przykłady:

```
CREATE VIEW ZestawienieUzytkownicy(Imie, Nazwisko, LiczbaFilmow, Saldo)
AS
SELECT Imie, Nazwisko, COUNT(DISTINCT Tytul), SUM(Cena)
FROM Wypozyczenia JOIN Uzytkownicy USING(IdUzytkownika)
JOIN Filmy USING(IdFilmu)
GROUP BY IdUzytkownika, Imie, Nazwisko
ORDER BY COUNT(DISTINCT Tytul) DESC, SUM(Cena) DESC;
```

```
CREATE VIEW ZestawienieFilmy(Tytul, LiczbaWypozyczen, Saldo)
AS
SELECT Tytul, COUNT(*), SUM(Cena)
FROM PelneWypozyczenia
GROUP BY Tytul
ORDER BY COUNT(*) DESC, SUM(Cena) DESC;
```

Ćwiczenie 3. Widok `ZestawienieFilmy` został utworzony z wykorzystaniem widoku `PelneWypozyczenia`. Czy zestawienie użytkowników można stworzyć w ten sam sposób?

Ćwiczenie 4. Stwórz widok zestawiający sumę podatków, jakie trzeba zapłacić od transakcji wykonanych w poszczególnych miesiącach. Wykorzystaj widok `PelneWypozyczenia`.

Ćwiczenie 5. Zastanów się czy w twoim projekcie zastosowanie widoków usprawni pracę z bazą. Jeżeli tak, stwórz odpowiednie widoki.

Procedury

Procedura jest blokiem instrukcji wykonywanych na bazie. Procedura może wczytywać pewne argumenty, ale nie zwraca wartości. Klasyczne problemy, które rozwiązujemy wykorzystują procedury, to m.in. dodawanie, usuwanie i aktualizowanie danych w tabelach oraz archiwizacja i przywracanie bazy.

Deklaracja najprostszej procedury ma następującą strukturę:

```
CREATE PROCEDURE NazwaProcedury(Arg1 Typ1, Arg2 Typ2, ...)
LANGUAGE język
AS $$
BlokInstrukcji (CiałoProcedury);
$$;
```

Jako języka procedury będziemy używać języka `SQL` lub `PLpgSQL`. W przypadku tego ostatniego blok procedur wpisujemy pomiędzy słowa kluczowe `BEGIN` i `END`. Można jednak pisać procedury również w innych językach, np. w C. Więcej o tworzeniu procedur i języku PL/pgSQL można przeczytać w dokumentacji.

Wywoływanie procedury odbywa się według następującego wzorca:

```
CALL NazwaProcedury(Arg1, Arg2, ...);
```

Przykład 3. Poniższa procedura usuwa wypożyczenia starsze niż zadany `Wiek` wyrażony w dniach.

```
CREATE PROCEDURE UsunStareWypozyczenia(Wiek INTEGER)
LANGUAGE SQL
AS $$
DELETE FROM Wypozyczenia
WHERE CURRENT_DATE-DataWypozyczenia > Wiek;
$$;
```

Następujące wywołanie procedury usunie wszystkie Wypozyczenia starsze niż rok.

```
CALL UsunStareWypozyczenia(365);
```

Więcej o tworzeniu procedur można przeczytać w dokumentacji.

Ćwiczenie 6. Zaproponuj procedury, które mogą usprawnić pracę z bazą w przygotowywanym projekcie.

Funkcje

Funkcje, podobnie jak procedury, są blokami instrukcji. Jednak przeciwieństwie do procedur, funkcje zwracają jakąś wartość (liczbę, napis, tabelę, itp.). Funkcje będziemy stosować do przetwarzania danych zależnych od pewnych zmiennych warunków, np. wyświetlanie danych i statystyk użytkowników czy zestawienia transakcji w nietypowych okresach czasu.

Schemat deklaracji funkcji jest bardzo podobny do deklaracji procedury:

```
CREATE FUNCTION NazwaFunkcji(Arg1 Typ1, Arg2 Typ2, ...) RETURNS Typ
AS $$
BlokInstrukcji (CiałoFunkcji);
$$ LANGUAGE język;
```

Tak stworzone funkcje działają jak wcześniej wspominane funkcje wierszowe. Wywołujemy je po słowie kluczowym `SELECT`.

Przykład 4. W języku PL/pgSQL możemy pisać klasyczne funkcje znane z kursów programowania, niezwiązane z bazą danych.

```
CREATE FUNCTION Dodaj(a NUMERIC, b NUMERIC) RETURNS NUMERIC
AS $$
BEGIN
   RETURN a+b;
END;
$$ LANGUAGE plpgsql;
```

Taką funkcję możemy wywołać w oderwaniu od bazy

```
SELECT Dodaj(2,5);
```

lub wykorzystać ją do przetwarzania danych pobieranych z bazy

```
SELECT Dodaj(Netto, Podatek)
FROM PelneWypozyczenia;
```

Innym klasycznym przykładem jest algorytm Euklidesa:

```
CREATE FUNCTION NWD(a INTEGER, b INTEGER) RETURNS INTEGER
AS $$
BEGIN
        WHILE a>0 AND b>0
        LOOP
        IF a>b THEN
        a:=a-b;
        ELSE
        b:=b-a;
        END IF;
        END LOOP;

        IF a=0 THEN
        RETURN b;
        ELSE
        RETURN a;
        END IF;
END;
$$ LANGUAGE plpgsql;
```

Uwaga. W powyższym przykładzie widzimy jak stosować pętlę `WHILE` i instrukcję warunkową `IF`.

Przykład 5. Poniżej mamy przykład funkcji, która wykorzystuje dane z bazy do stworzenia bilansu dla wybranego filmu, identyfikowanego przez tytuł i rok produkcji.

```
CREATE FUNCTION SaldoFilmu(TytulFilmu TEXT, RokFilmu INTEGER)
RETURNS NUMERIC
AS $$

DECLARE Saldo NUMERIC;

BEGIN
Saldo := (SELECT SUM(Cena)
          FROM Filmy JOIN Wypozyczenia USING(IdFilmu)
          WHERE Tytul = TytulFilmu AND Rok = RokFilmu);
RETURN Saldo;
END;
$$ LANGUAGE plpgsql;
```

Jeżeli w ciele funkcji chcemy korzystać ze zmiennych pomocniczych, musimy je najpierw zadeklarować po słowie kluczowym `DECLARE`.

Podobnie jak procedury, naszą funkcję możemy wywołać dla pojedynczego filmu

```
SELECT SaldoFilmu('Amistad', 2000);
```

lub dla serii filmów wypisanych w tabeli

```
SELECT Tytul, Rok, SaldoFilmu(Tytul, Rok)
FROM Filmy;
```

Ćwiczenie 7. Zastanów się czy w przygotowywanym projekcie występują problemy, których rozwiązania można usprawnić za pomocą funkcji. Jeżeli tak, to napisz odpowiednie funkcje.

Wyzwalacze

Szczególnym typem procedur są wyzwalacze. Wywołują się one automatycznie przy wykonywaniu modyfikacji wpisów do bazy, tj. zapytań typu `INSERT`, `UPDATE` oraz `DELETE`. Wyzwalacz wywołuje tzw. funkcję wyzwalaczową, która zawsze zwraca wartość typu `TRIGGER`. Może ona również korzystać ze zmiennych związanych z wyzwalaczem jak np. operacja, która wywołała wyzwalacz czy modyfikowane dane.

Wyzwalacze deklarujemy dwustopniowo – najpierw deklarujemy funkcję wyzwalaczową, a potem tworzymy właściwy wyzwalacz, który ją wywołuje. Tworzenie wyzwalacza przebiega według następującego schematu:

```
CREATE FUNCTION NazwaFunkcji() RETURNS trigger
AS $NazwaFunkcji$
CiałoFunkcji;
$NazwaFunkcji$ LANGUAGE język;

CREATE TRIGGER Nazwa [BEFORE|AFTER|INSTEAD OF] Operacja
ON NazwaTabeli
[FOR EACH ROW]
EXECUTE PROCEDURE NazwaFunkcji();
```

Jeżeli funkcja wyzwalaczowa operuje na wierszach tabeli, to przed jej wywołaniem musimy dopisać `FOR EACH ROW` (tzw. wyzwalacz wierszowy). Możliwymi operacjami są `INSERT`, `UPDATE`, `DELETE` oraz ich kombinacje.

Przykład 6. Prostym przykładem wyzwalacza może być procedura zabraniająca usuwania zarejestrowanych wypożyczeń.

```
CREATE FUNCTION NieUsuwajWypozyczen() RETURNS trigger
AS $NieUsuwajWypozyczen$
BEGIN
RAISE EXCEPTION 'Nie można usuwać wypozyczeń.';
END;
$NieUsuwajWypozyczen$ LANGUAGE plpgsql;

CREATE TRIGGER NieUsuwajWypozyczen BEFORE DELETE
ON Wypozyczenia
EXECUTE PROCEDURE NieUsuwajWypozyczen();
```

Wyjątek (`RAISE EXCEPTION`) przerywa operację i wyświetla odpowiedni komunikat.

Przykład 7. Stworzymy teraz wyzwalacz, który nie pozwoli użytkownikowi wprowadzać filmów, które jeszcze się nie ukazały (rok wydania jest późniejszy niż obecny).

```
CREATE OR REPLACE FUNCTION RokFilmu() RETURNS trigger
AS $RokFilmu$
BEGIN
IF NEW.Rok > EXTRACT(YEAR FROM CURRENT_DATE) AND TG_OP = 'INSERT' THEN
RAISE EXCEPTION '% > %: Nie można dodawać filmów z przyszłości',
                NEW.Rok, EXTRACT(YEAR FROM CURRENT_DATE);
END IF;

IF NEW.Rok > EXTRACT(YEAR FROM CURRENT_DATE) AND TG_OP = 'UPDATE' THEN
RAISE EXCEPTION '% > %: Nie można przenosić filmów do przyszłości',
                NEW.Rok, EXTRACT(YEAR FROM CURRENT_DATE);
END IF;
RETURN NEW;
END;
$RokFilmu$ LANGUAGE plpgsql;

CREATE TRIGGER RokFilmu BEFORE INSERT OR UPDATE
ON Filmy
FOR EACH ROW
EXECUTE PROCEDURE RokFilmu();
```

W treści komunikatu wywołanego wyjątku mogą pojawiać się wartości zmiennych. Oznaczamy je znakiem `%` i zmienne w odpowiedniej kolejności wypisujemy po przecinku za komunikatem. Specjalna zmienna `TG_OP` zawiera nazwę akcji, która aktywowała wyzwalacz. Natomiast w zmiennej `NEW` mamy zapisaną wprowadzaną krotkę. Więcej o specjalnych zmiennych można przeczytać w dokumentacji.

Uwaga. Odpowiednio przemyślane wyzwalacze mogą bardzo usprawnić pracę z bazą. Jednak ich nadmiar lub nieprzemyślana struktura mogą uniemożliwić niektóre operacje. Na przykład blokada usuwania wypożyczeń – z jednej strony wydaje się być sensowna, gdyż wypożyczalnie VOD najczęściej nie uznają zwrotów. Co więcej warto się zabezpieczyć przed utratą danych historycznych. Z drugiej jednak strony taki wyzwalacz uniemożliwia archiwizację (przeniesienie danych do innej tabeli).

Ćwiczenie 8. Usuń wyzwalacz z przykładu 6. Stwórz tabelę `Archiwum` i napisz wyzwalacz, który przy dowolnej operacji na tabeli `Wypozyczenia` wykonana archiwizację transakcji starszych niż 180 dni (skopiuje stare wpisy do `Archiwum` i usunie je z tabeli `Wypozyczenia`). Zabezpiecz tabelę `Archiwum` przed usuwaniem danych.

Ćwiczenie 9. Zaproponuj wyzwalacze, które usprawnią pracę z bazą w przygotowywanym projekcie. Pamiętaj, by zaprogramowane funkcje nie utrudniały pracy z bazą i nie uniemożliwiały koniecznych operacji.

Odnośniki i załączniki

[1] Dokumentacja języka PL/pgSQL: https://www.postgresql.org/docs/current/plpgsql.html