MASARYKOVA UNIVERZITA
INSTITUT BIOSTATISTIKY A ANALÝZ

Databázové systémy a SQL

Lekce 11

Daniel Klimeš

Uživatelské funkce

- Objekty databáze, stejně jako tabulky
- Vytvoření příkazem CREATE, zrušení příkazem DROP
- Možné sdílení mezi uživateli, lze definovat oprávnění na spuštění
- Skládá se z DML SQL příkazů
- Konstrukce jazyka **PL/pgSQL** – Procedural Language
- Návratová hodnota
- Použití SELECT funkce()

Základy PL/pgSQL

- Standardní procedurální programovací jazyk, obdoba C, Java, Pascal
- Příkazy se vykonávají postupně + programovací smyčky
- Průchod tabulkou řádek po řádku

Základní prvky

- Bloky kódu ohraničeny BEGIN END
- Definice proměnných
- Operátor přiřazení hodnoty do proměnné
- Podmíněný výraz
- Programovací smyčka
- Volání jiných funkcí
- Prvky odděleny středníkem

PL/SQL - FOR

```
DECLARE
   rs RECORD; --deklarace kurzoru
BEGIN
FOR rs IN (SELECT * FROM patients) LOOP
          IF (rs.date_of_birth > current_date) THEN
                    INSERT INTO test_tab (patient_id) VALUES (rs.patient_id);
          END IF; -- ukončení podmíněného výrazu
END LOOP; -- ukončení smyčky
END;
```

- FOR rs IN (SELECT * FROM patients) LOOP
  - Příkaz smyčky
  - Proměnná rs (kurzor, „vektor“) postupně nabývá hodnot řádků, které vrací SELECT příkaz (jednotlivé pacienty)
  - Kurzor rs se musí deklarovat jako RECORD
  - Pro každý vrácený řádek SELECT příkazu se provedou příkazy uzavřené mezi LOOP a END LOOP
  - Smyčka končí po zpracování všech záznamů SELECTU
  - Pokud SELECT nevrací žádné řádky, blok smyčky se přeskočí

PL/SQL - IF

```
BEGIN
FOR rs IN (SELECT * FROM patients) LOOP
        IF (rs.date_of_birth > current_date) THEN
                INSERT INTO test_tab (patient_id) VALUES (rs.patient_id);
        END IF; -- ukončení podmíněného výrazu
END LOOP; -- ukončení smyčky
END;
```

- IF (rs.date_of_birth > current_date) THEN
  - podmíněný výraz
  - pokud je splněna podmínka, provedou se příkazy mezi THEN a END IF
  - Pokud ne, pokračuje se až za END IF

Proměnná v PL/SQL

```
CREATE OR REPLACE FUNCTION testf()
RETURNS NUMERIC AS $$
DECLARE
   rs RECORD;
   i NUMERIC(3);
BEGIN
i:=0;
FOR rs IN (SELECT * FROM patients) LOOP
            IF (rs.date_of_birth > current_date) THEN
                        INSERT INTO test_tab (patient_id) values (rs.patient_id);
                        i:=i+1;
            END IF; -- ukončení podmíněného výrazu
END LOOP; -- ukončení smyčky
RETURN i;
END;
$$ LANGUAGE PLPGSQL;
```

- DECLARE – zahajuje blok definice proměnných, každá proměnná musí být deklarovaná na začátku kódu

Operátor přiřazení – :=

Výjimky

```
CREATE OR REPLACE FUNCTION testf()
RETURNS NUMERIC AS $$
DECLARE
   rs RECORD;
   i NUMERIC(3);
BEGIN
i:=0;
FOR rs IN (SELECT * FROM patients) LOOP
            IF (rs.date_of_birth > current_date) THEN
                        INSERT INTO test_tab (patient_id) values (rs.patient_id);
                        i:=i+1;
            END IF; -- ukončení podmíněného výrazu
END LOOP; -- ukončení smyčky
RETURN i;
EXCEPTION
   WHEN division_by_zero THEN - -konkrétní očekávaná chyba
      RAISE NOTICE 'Neumim delit nulou';
      RETURN i;
   WHEN OTHERS THEN  - - další chyby
       RAISE NOTICE 'Neco je spatne: %', SQLERRM;
       RETURN i;
END;
$$ LANGUAGE PLPGSQL;
```

Procedura - vytvoření časové osy

Přehled počtu zařazených pacientů po měsících:

```
CREATE VIEW mesicni_pocty AS
SELECT  TO_CHAR(date_of_enrollment, 'yyyy-mm') mesic, COUNT(*) pocet
FROM
patient_study WHERE study_id = 43
GROUP BY  TO_CHAR(date_of_enrollment, 'yyyy-mm')
ORDER BY 1
```

Chybí některé měsíce → **Vytvoření časové osy v pomocné tabulce**

- Tabulka KALENDAR, její naplnění funkcí PROC_KALENDAR
- CREATE TABLE kalendar (
  Mesic VARCHAR(10)
)

PL/SQL procedura

```
CREATE OR REPLACE FUNCTION proc_kalendar(od DATE, mesicu NUMERIC)
RETURNS NUMERIC AS $$
DECLARE
   i NUMERIC(3);
BEGIN
DELETE FROM kalendar;
  FOR i IN 0..mesicu-1 LOOP
    INSERT INTO kalendar (mesic) VALUES (to_char(od + (interval '1 month' * i), 'yyyy-mm'));
  END LOOP;
RETURN i;
EXCEPTION
   WHEN OTHERS THEN
       RAISE NOTICE 'Neco je spatne: %', SQLERRM;
       RETURN -1;
END;
$$ LANGUAGE PLPGSQL;
```

PL/SQL procedura

Spuštění:

SELECT proc_kalendar ('2010-01-01', 24);

Doplněný výpis:

```
SELECT k.mesic, COALESCE(mp.pocet,0) pocet FROM
kalendar k LEFT JOIN mesicni_pocty mp ON k.mesic = mp.mesic
ORDER BY k.mesic
```

POSTGRESQL – generátor řádků

```
SELECT a FROM GENERATE_SERIES(1,100) a ;
```

Tabulky

- PostgreSQL (ANSI standard)
  - information_schema.tables
  - information_schema.columns

DDL příkazy pro manipulaci s tabulkami
- CREATE TABLE
- DROP TABLE
- ALTER TABLE

Pohledy (VIEWS)

•Pohled = uložený SQL dotaz
•Pracuje se s ním stejně jako s tabulkou
•Ve většině případů je možný pouze SELECT

• CREATE VIEW v_ukazka AS
SELECT ps.patient_id, study_name FROM patient_study ps, studies s
WHERE ps.study_id = s.stud y_id

SELECT study_name, count(*) FROM v_ukazka
GROUP BY study_name

DDL pro pohledy:
CREATE OR REPLACE VIEW AS
DROP VIEW

PostgreSQL/ANSI
• information_schema.views

Indexy

- Indexy jsou obdobou kartotéky
- Umožňují rychlejší vyhledávání záznamů ve velkých tabulkách
- Urychlují SELECT dotazy, zpomalují INSERT, UPDATE, DELETE

- Indexy se vytváří nad jedním nebo více sloupci tabulky
- Standardně nad primárním klíčem a cizími klíči
- Dále nad sloupci, které se často používají za WHERE

- DDL pro indexy
  - CREATE INDEX
  - DROP INDEX
  - ALTER INDEX

Sekvence

- Sekvence generují za všech okolností unikátní čísla – posloupnost
- Použití pro primární klíče při insertech nových řádků

- SELECT NEXTVAL(‘nazev_sekvence')
- SELECT CURRVAL('nazev_sekvence ')

- Každé zavolání NEXTVAL vrátí další číslo v posloupnosti bez ohledu na transakce
- Při neúspěšném použití vygenerovaného ID vznikají “díry” v posloupnosti

- DDL
  - CREATE SEQUENCE
  - DROP SEQUENCE
- PostgreSQL/ANSI
  - information_schema.sequences