Datové a procesní modely

Relační databáze

Uložené procedury/funkce

Jiří Měska

jiri.meska@silicium.cz

MFFUK Praha

2020

SQL - neprocedurální jazyk

sql povel

```
select * from osoba;
```

sql skript

```
/* ---------------------  */
/* Založení DB Mzdy  */
/* ---------------------  */
Create Table Osoba (
      Id  Serial Prim..Key,
      Jm VarChar(30),
      Pr   VarChar(30),
                  ..... );
Create Table  Odd (
            .....
            );
 ……
```

Textový soubor.

Soubor se zkratkou ____.sql

Provádí se povel po povelu.

Pokud všechny povely dobře, provede se celý skript.

Pokud nějaký povel chybný přeruší se jím skript.

Script je uložená sequence povelů.

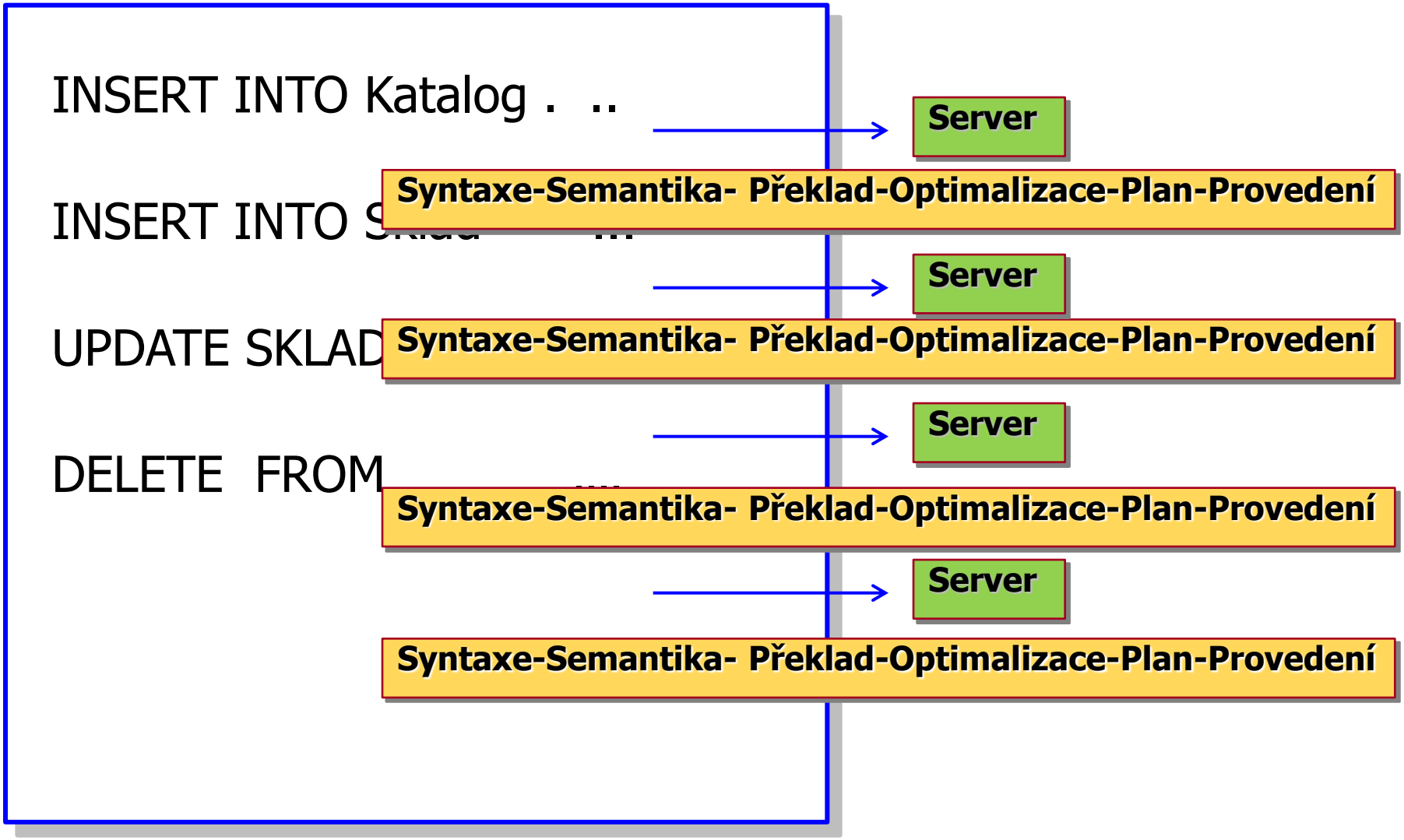
i.Relační databáze je však také vybavena standardním procedurálním jazykem

i.Každý výrobce SQL má vlastní procedurální jazyk.

ii.Starndard SQL se nazývá **SQL/PSM**

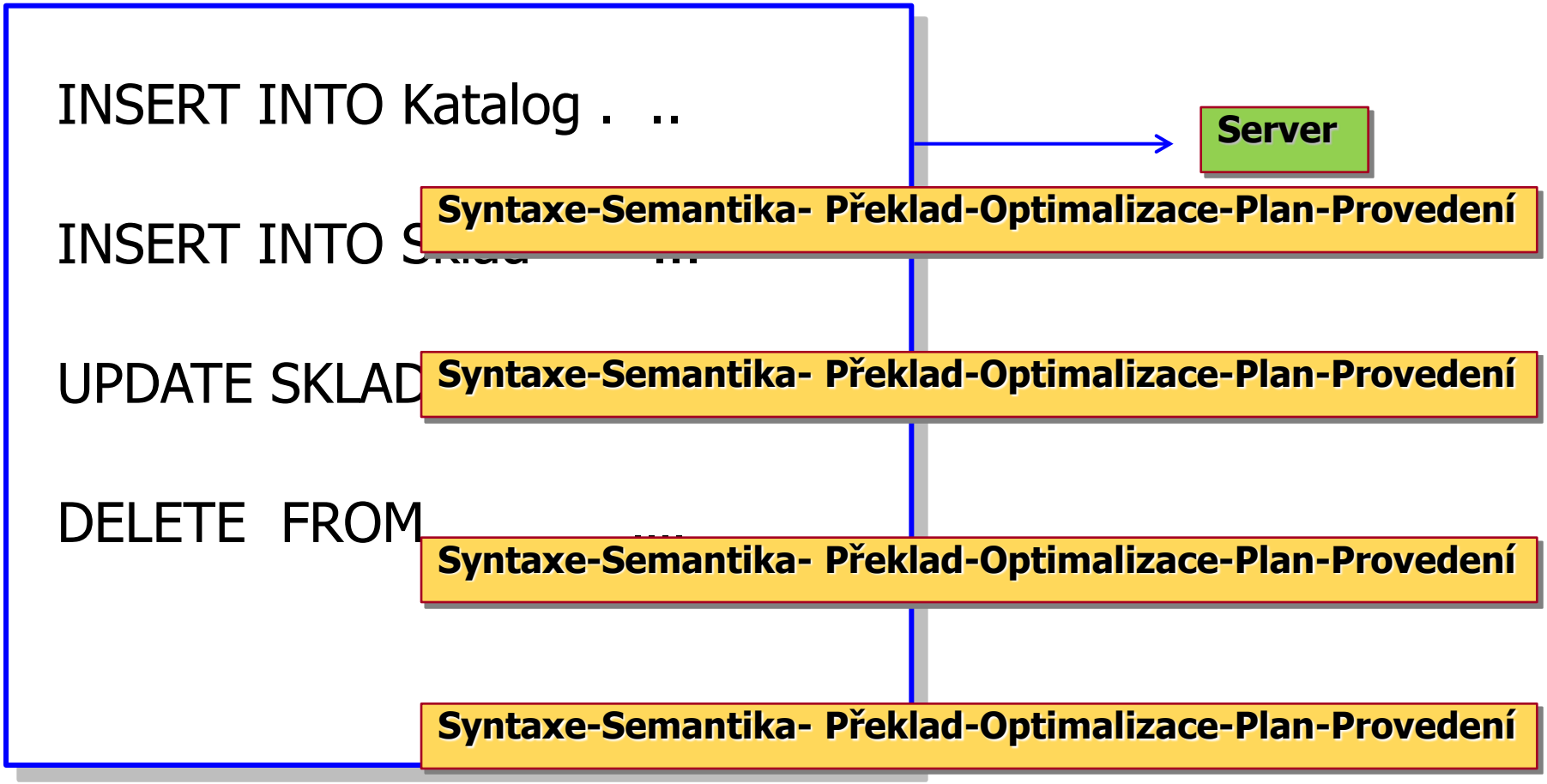
SQL – neprocedurální jazyk

Posloupnost povelů SQL zpracovávaných jeden po druhém



SQL – neprocedurální jazyk

SQL Script - zpracuje se najednou



SQL/PSM - procedurální jazyk SQL databází

- SQL/PSM – standard SQL [ANSI-ISO]
  - IBM DB2
  - MySQL
- Vlastní proprietární rozšíření – deriváty normy - objevují se dříve než norma
  - Oracle PL/SQL
  - PostgreSQL PL/pgSQL
  - Microsoft T-SQL
  - SyBase T-SQL
  - .....

a něco uloženého a jenom spustitelného

Uložené úlohy

```
INSERT INTO Katalog ...
INSERT INTO Sklad   ...
UPDATE SKLAD SET    ...
DELETE  FROM
....
```

```
INSERT INTO Katalog ...
INSERT INTO Sklad   ...
UPDATE SKLAD SET    ...
DELETE  FROM
....
```

```
INSERT INTO Katalog ...
INSERT INTO Sklad   ...
UPDATE SKLAD SET    ...
DELETE  FROM
....
```

- provádění stejné [podobné] sekvence povelů
- provádění ucelené jednotky práce

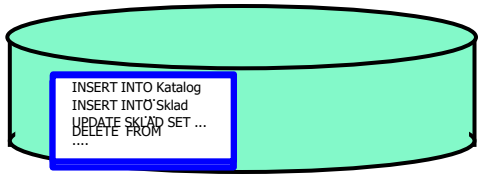
Uložené úlohy

CALL ...

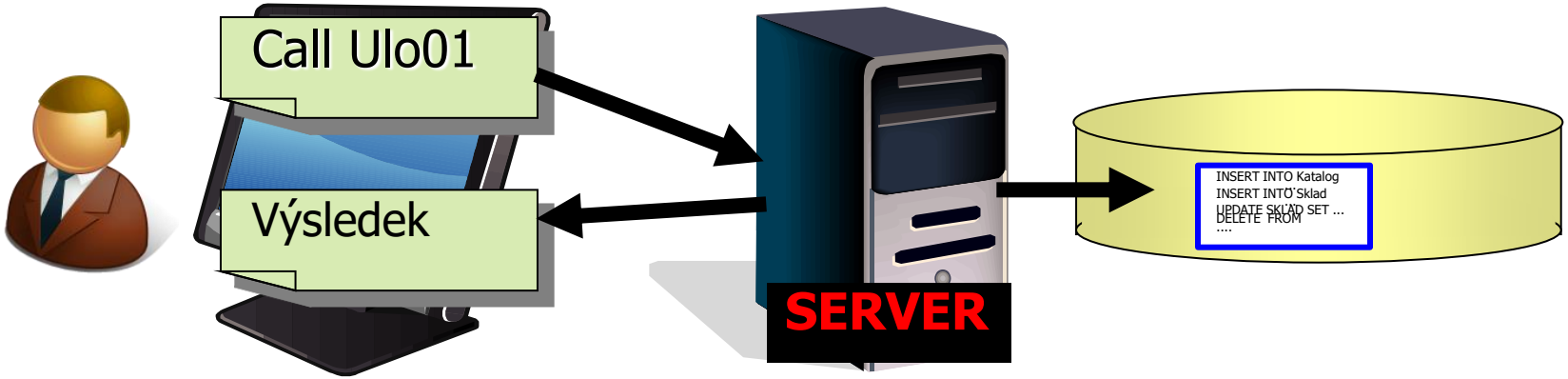
CALL ...

CALL ...

- Uložená úloha se aktivuje povelem **CALL ()** nebo **EXECUTE()**
- Uložená úloha může být parametrizována



Uložená úloha



- sada povelů SQL
- uložená jako jedna úloha na serveru
- úloha pojmenovaná
- úloha zkompilovaná [pro rychlejší použití]
- vyvolávaná a prováděná na serveru
- (metody DB serveru)
- uložená úloha je **databázový objekt**

Co uložené procedury umí

- deklarace proměnných
- deklarace kurzorů - výsledek dotazu
- řízení toku zpracování

  smyčky, podmínky
- ošetření zvláštních stavů [chybových]
- vzájemné volání procedur

Co uložené procedury umožňují

- umožňují vytvářet knihovny procedur pro konzistentní práci s databází
- umožňují sjednotit logiku práce více aplikací do společného jádra
- mezivýsledky při běhu uložené procedury není třeba přenášet mezi serverem a klientem, tím se šetří čas zpracování i využití paměti jak na databázovém stroji, tak u klienta
- databázový stroj překládá uloženou proceduru jednou a tím se šetří čas zpracování na databázovém stroji

Opakování: Co je to spouštěč? (trigger)

- automaticky spuštěné povely u změnových SQL operaci
  [**Insert, Update, Delete**]
- Mohou být spuštěny
- PŘED provedením změny ..... **Before**
- PO provedení změny ..... **After**
- MÍSTO provedení změny ..... **Instead Of**

- jednou pro daný povel (událost) ..... **For Each Statement**
- či není-li zadáno .....
- jednou pro každou řádku ..... **For Each Row**

  vkládanou (Insert)

  měněnou (Update)

  rušenou (Delete)

PL/PgSQL - jazyk procedur v PostgreSQL - odkazy

- https://www.tutorialspoint.com/postgresql/postgresql_functions.htm
- https://w3resource.com/PostgreSQL/tutorial.php
- https://www.w3resource.com/PostgreSQL/pl-pgsql-tutorial.php

- PL/PgSQL je procedurální programovací jazyk PostgreSQL
- Je podobný Oracle PL/SQL
- Umožňuje vytvářet funkce a spoušťěče (trigger)
- Rozšiřuje SQL o řídící příkazy běhu zpracování

PL/PgSQL - jazyk procedur v PostgreSQL - příklad

```
-- ============================        --
-- Příklad vytvoření funkce  - počet adres        --
-- ============================        --
CREATE OR REPLACE FUNCTION totalRecords ()
RETURNS integer AS $total$
declare
    total integer;
BEGIN
   SELECT count(*) into total FROM adresa;
   RETURN total;
END;
$total$ LANGUAGE plpgsql;
```

```
-- ==============   --
-- Příklad volání funkce    --
-- ==============   --
select totalRecords ();
```

PL/PgSQL - Struktura jazyka

PL/pgSQL je blokově strukturován a každý příkaz uvnitř bloku je ukončen středníkem.
Pokud je uveden label ($total$), musí být na začátku i na konci.

```
CREATE [OR REPLACE] FUNCTION function_name (arguments)
RETURNS return_datatype AS $total$
  DECLARE
    declaration;
    [...]
  BEGIN
    < function_body >
    [...]
    RETURN { variable_name | value }
  END;
$total$  LANGUAGE plpgsql;
```

PL/PgSQL - Deklarace proměnných

```
name [ CONSTANT ] type [ COLLATE collation_name ] [ NOT NULL
    ] [ { DEFAULT | := } expression ];
```

id integer; -- proměnná typu integer

pocet numeric(5); -- proměnná typu numerického pole

description varchar; -- textová proměnná

radka tablename%ROWTYPE; -- lze načíst řádku tabulky

sloupec tablename.columnname%TYPE; -- lze načíst sloupce zadané tabulky

sloupec RECORD; -- lze načíst sloupce libovolné tabulky

```
id integer  DEFAULT 1;
moje CONSTANT integer := 10;
url varchar := 'http://example.com';
```

PL/PgSQL - Pomocný výpis do záložky Messages

```
CREATE OR REPLACE FUNCTION totalRecords () RETURNS integer AS $$
DECLARE
        total integer;
        tabl adresa%ROWTYPE;
BEGIN
   SELECT count(*) into total FROM adresa;
   SELECT * into tabl FROM adresa where idadresa=4001;
   RAISE NOTICE 'Debug: % vypis proměnné', total;
   RAISE NOTICE 'Debug: % vypis proměnné', tabl;
    RETURN total;
END;
$$ LANGUAGE plpgsql;
```

Data Output | Explain | Messages | Notifications | Query History

	totalrecords integer
1	12

Data Output | Explain | Messages | Notifications | Query History

```
NOTICE:  Debug: 12 vypis proměnné
NOTICE:  Debug: (4001,Praha,"Kavkova 324",6001) vypis proměnné

Successfully run. Total query runtime: 68 msec.
1 rows affected.
```

Pomocný výpis se objeví v pgAdmin v záložce Messages
Výsledek se objeví v pgAdmin v záložce Messages Data Output

PL/PgSQL– funkce s parametry a její volání

```
CREATE OR REPLACE FUNCTION sum_of_two_numbers(
    m integer, n integer)
RETURNS integer AS $$
        BEGIN
                RETURN m + n;
        END;
$$ LANGUAGE plpgsql;
```

```
Select sum_of_two_numbers(5,6) ;
```

Poznámka: CREATE OR REPLACE umožní změnit funkci se stejnou typovou strukturou parametrů.

Pokud chceme změnit typ parametrů nebo jejich strukturu, je třeba funkci nejprve zrušit:

DROP FUNCTION sum_of_two_numbers ();

PL/PgSQL– funkce s parametry a její volání

Na parametry se lze odvolávat i přes pozice parametrů: $1, $2

Modifikace předchozího příkladu:

```
CREATE OR REPLACE FUNCTION sum_of_two_numbers(integer, integer)
RETURNS integer AS $$
        BEGIN
                RETURN $1 + $2;
        END;
$$ LANGUAGE plpgsql;
```

```
Select sum_of_two_numbers(5,6) ;
```

PL/PgSQL - volání funkce s/bez návratové hodnoty

```
CREATE OR REPLACE FUNCTION test() RETURNS void AS $$
DELETE FROM cislo;
INSERT INTO cislo VALUES (31),(51)
$$ LANGUAGE sql;

CREATE OR REPLACE FUNCTION demo() RETURNS integer
AS $$
DECLARE
    ret_value integer := 0;
BEGIN
        ret_value = sum_of_two_numbers(13,17);
        PERFORM test();
RETURN ret_value;
END; $$ LANGUAGE plpgsql;
```

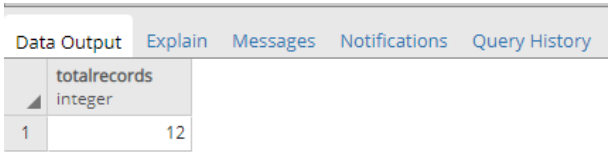
```
select demo ();
```

Data Output | Explain | Messages | Notifications | Query History

	demo integer
1	30

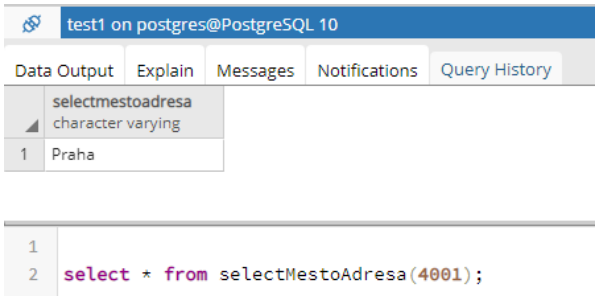
PL/PgSQL - Výpis řádků tabulky

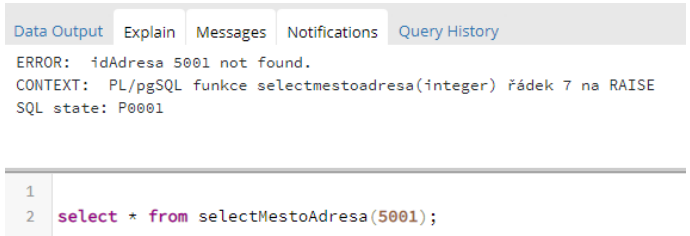
```
CREATE OR REPLACE FUNCTION totalRecords ()
RETURNS integer AS $$
declare
        total integer default 0;
        radka record;
BEGIN
   FOR radka IN SELECT * FROM adresa LOOP
      RAISE NOTICE 'Výpis řádku: %', radka ;
      total := total + 1;
   END LOOP;
   RETURN total;
END;
$$ LANGUAGE plpgsql;
```



PL/PgSQL - Exception

```
CREATE OR REPLACE FUNCTION selectMestoAdresa ( idAdr integer )
RETURNS varchar AS $BODY$
declare
  mesto1 varchar;
BEGIN
  SELECT mesto INTO mesto1 FROM adresa  where idAdresa = idAdr;
  IF NOT FOUND THEN
                RAISE EXCEPTION 'idAdresa % not found.', idAdr;
  END IF;
  RETURN mesto1;
END $BODY$ LANGUAGE plpgsql;
```






```
ERROR:  idAdresa 5001 not found.
CONTEXT:  PL/pgSQL funkce selectmestoadresa(integer) řádek 7 na RAISE
SQL state: P0001
```

PL/PgSQL– dynamicky vytvořený příkaz (Execute)

```
CREATE OR REPLACE FUNCTION totalRecords ( tabulka varchar)
RETURNS integer AS $total$
declare
        total integer;
BEGIN
   EXECUTE 'SELECT count(*) FROM ' || tabulka INTO  total;
   RETURN total;
END;
$total$ LANGUAGE plpgsql;
```

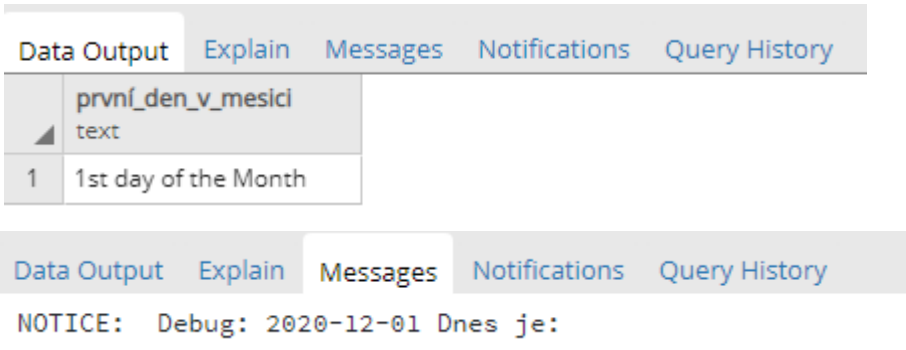
```
select * from totalRecords ( 'obchodnik' );
```

Data Output | Explain | Messages | Notifications | Query History

	totalrecords integer
1	9

PL/PgSQL– Větvení programu IF-THEN-ELSE

```
CREATE OR REPLACE FUNCTION první_den_v_mesici()
RETURNS text
AS $$
BEGIN
   RAISE NOTICE 'Debug: % Dnes je: ', current_date;
   IF EXTRACT(DAY FROM current_date) = 1
   THEN
      RETURN '1st day of the Month';
   ELSE
      RETURN 'Other day';
   END IF;
END;
$$ LANGUAGE plpgsql;
```



PL/PgSQL– Větvení programu IF-THEN-ELSE

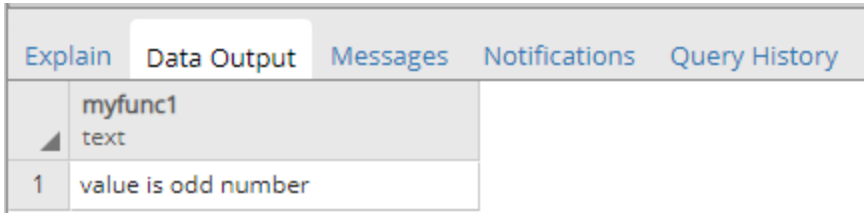
```
CREATE OR REPLACE FUNCTION seazon (crdate date) RETURNS
text AS $$
BEGIN
   IF EXTRACT (MONTH FROM crdate) in (1,2,3)
   THEN RETURN 'It is winter';
   ELSEIF EXTRACT (MONTH FROM crdate) in (4,5,6)
   THEN RETURN 'It is spring';
   ELSEIF EXTRACT (MONTH FROM crdate) in (7,8,9)
   THEN RETURN 'It is summer';
   ELSEIF EXTRACT (MONTH FROM crdate) in (10,11,12)
   THEN RETURN 'It is autumn';
 END IF;
END;
$$ LANGUAGE plpgsql;
```

```
select * from
seazon (to_date('2020/12/1','yyy/mm/dd'));
```

PL/PgSQL - CASE

```
CREATE OR REPLACE FUNCTION myfunc1 (x integer) RETURNS
text AS $$
DECLARE
msg text; rest integer := x % 10;
BEGIN
CASE
    WHEN rest IN(0,2,4,6,8) THEN    msg := 'value even number';
    WHEN rest IN(1,3,5,7,9) THEN    msg := 'value is odd number';
    ELSE msg := 'nenastane';
END CASE;
RETURN msg;
END;
$$ LANGUAGE plpgsql;
```

```
select * from myfunc1(777);
```



PL/PgSQL– LOOP with EXIT WHEN

```
LOOP
    statement;
    [...]
    EXIT [ label ] [ WHEN condition ];
 END LOOP;
```

```
cub := nm;
LOOP
      cub := cub+1;
      EXIT WHEN cub >= 10000;
END LOOP;
```

PL/PgSQL– LOOP with WHILE

```
WHILE condition LOOP
statement;
[...]
END LOOP;
```

```
cub:=nm;
WHILE cub <=10000 LOOP
  cub := cub+1;
END LOOP
```

PL/PgSQL– Trigger

An event could be any of the following: INSERT, UPDATE, DELETE or TRUNCATE.

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF} {event [OR ...]}
   ON table_name
   [FOR [EACH] {ROW | STATEMENT}]
       EXECUTE PROCEDURE trigger_function
```

PL/PgSQL– Create audit table for table adresa

```
CREATE TABLE adresa_audit (
   id SERIAL PRIMARY KEY,
   idAdresa INT NOT NULL,
   last_mesto VARCHAR(40) NOT NULL,
   changed_on TIMESTAMP(6) NOT NULL
)
```

Vytvoření auditovací tabulky pro změny v tabulce Adresa

PL/PgSQL– Trigger function

Vytvoříme funkci pro aktualizační trigger, zapíše do auditovací tabulky

```
CREATE OR REPLACE FUNCTION adresa_audit_triger()
  RETURNS trigger AS $BODY$
BEGIN
   IF NEW.mesto <> OLD.mesto THEN
      INSERT INTO adresa_audit(idAdresa,last_mesto,changed_on)
      VALUES(OLD.idadresa,OLD.mesto,now());
   END IF;
   RETURN NEW;
END;
$BODY$ LANGUAGE plpgsql;
```

PL/PgSQL– Trigger for audit changes in table adresa

```
CREATE TRIGGER audit_adresa AFTER INSERT OR UPDATE OR
DELETE ON adresa
FOR EACH ROW EXECUTE PROCEDURE adresa_audit_triger();
```

```
update Adresa set mesto = 'Pardubice' where idadresa = 4001;
```

```
select * from adresa_audit;
```

Explain | Data Output | Messages | Notifications | Query History

	id integer	idadresa integer	last_mesto character varying (40)	changed_on timestamp without time zone
1	1	4001	Praha	2020-12-01 21:32:43.755086

PL/PgSQL– Drop trigger

Zrušení trigger lze udělat příkazem drop

drop trigger <trigger_name> on <table_name>;

PL/PgSQL– Rozlišení běhu trigger pro insert, update

Někdy se může hodit rozlišit, jestli procedura běží pro insert nebo update.

Následující ukázka z procedury:

```
IF (TG_OP = 'INSERT') THEN oldDatumZaplaceni = null; END IF;

IF (TG_OP = 'UPDATE') THEN oldDatumZaplaceni =
OLD.datumzaplaceni; END IF;
```