1. Funkcje.

W systemie bazy danych PostgreSQL udostępniono definiowanie własnych funkcji oraz procedur. Do tego celu wykorzystuje się język SQL oraz częściej język pl/pgSQL. Istnieje możliwość skorzystania z innych języków takich jak C, python czy perl. Jeżeli korzystamy z innego języka niż SQL i pl/pgSQL mamy przygotować odpowiednie biblioteki i dołączyć do bazy danych oraz odpowiednią formą polecenia CREATE FUNCTION utworzyć obiekty funkcji w bazie danych. Od wersji 11 PostgreSQL udostępniono polecenie CREATE PROCEDURE, które ma podobne zadania do CREATE FUNCTION. Dwie zasadnicze różnice to:
- funkcje zwracają wyniki, procedury tego nie potrafią,
- procedury wspierają transakcje, funkcje nie.
Zasadniczym celem wprowadzenia tych obiektów jest modularyzacja kodu, wielokrotne korzystanie z raz utworzonego kodu, rozszerzenie możliwości bazy danych, udostępnienie interfejsu dla aplikacji baz danych.

Składnia polecenia dla funkcji.

https://www.postgresql.org/docs/9.1/sql-createfunction.html

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | WINDOW
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | COST execution_cost
    | ROWS result_rows
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
    [ WITH ( attribute [, ...] ) ]
```

Składnia polecenia dla procedury.

https://www.postgresql.org/docs/11/sql-createprocedure.html

```
CREATE [ OR REPLACE ] PROCEDURE
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
  { LANGUAGE lang_name
    | TRANSFORM { FOR TYPE type_name } [, ... ]
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
  } ...
```

Przykłady na laboratorium.

Procedury składowane w języku SQL.

```
-- Funkcja wyświetlająca uczestników określonego kursu - realizacja zapytania parametrycznego.
CREATE OR REPLACE FUNCTION sql1 ( ikurs int )
RETURNS SETOF uczestnik AS
$$
   SELECT u.id_uczestnik, u.nazwisko, u.imie
       FROM uczestnik u JOIN uczest_kurs uk ON u.id_uczestnik = uk.id_uczest
       WHERE uk.id_kurs = ikurs;
$$
LANGUAGE SQL;

SELECT sql1(1) ;        -- funkcja w obszarze argumentów
SELECT * FROM sql(1);    -- funkcja w obszarze źródeł danych ( po FROM )

DROP FUNCTION sql1(int); -- usunięcie funkcji sql1(int)
```

```
-- od wersji 11
CREATE OR REPLACE PROCEDURE uczestnik_dane ( id int )
LANGUAGE plpgsql

AS
$$
DECLARE
u_data varchar;
sep varchar DEFAULT ';';
BEGIN
```

```
   SELECT CONCAT( u.id_uczestnik::varchar, sep, u.nazwisko, sep, u.imie) INTO
u_data
        FROM uczestnik u
        WHERE id_uczestnik = id;
   RAISE NOTICE 'Dane uczestika % : %', id, u_data;
END;
$$;

--call uczestnik_dane (2);

--DROP PROCEDURE uczestnik_dane ( int );
```

```
--Deklaracja typu zwracanych wyników, z parametrem i bez parametru SETOF.

CREATE OR REPLACE FUNCTION lab04.getone ( int, int )
RETURNS lab04.uczestnik AS
$body$
  SELECT * FROM lab04.uczestnik WHERE id_uczestnik BETWEEN $1 AND $2 ;
$body$
LANGUAGE SQL;

CREATE OR REPLACE FUNCTION lab04.getmany ( int, int )
RETURNS SETOF lab04.uczestnik AS
$body$
  SELECT * FROM lab04.uczestnik WHERE id_uczestnik BETWEEN $1 AND $2 ;
$body$
LANGUAGE SQL;

-- sprawdzenie działania

SELECT * FROM lab04.getone(1,5);
SELECT * FROM lab04.getmany(1,5);

-- usunięcie
DROP FUNCTION lab04.getone(INT,INT);
DROP FUNCTION lab04.getmany(INT,INT);
```

Procedury składowane w języku PL/pgSQL.

Konstrukcja pętli.

```
---- LOOP ------------
DO $$
DECLARE
 i INTEGER := 0;
BEGIN
 LOOP
  EXIT WHEN i>9;
  i := i + 1;
  RAISE NOTICE 'i: %',i;
 END LOOP;
END; $$ ;

DO $$
DECLARE
 i INTEGER := 0;
BEGIN
 WHILE i<10 LOOP
  i := i + 1;
  RAISE NOTICE 'i: %',i;
 END LOOP;
END; $$;

DO $$
BEGIN
 FOR i IN 1..10 LOOP
  RAISE NOTICE 'i: %',i;
 END LOOP;
END; $$;
```

```
CREATE FUNCTION lab04.silnia (n integer)
RETURNS BIGINT AS
$$                -- otwarcie bloku programowego
  DECLARE
  i INTEGER := 0;
  sil BIGINT := 1;
BEGIN
  LOOP
    EXIT WHEN i>=n;
    i := i + 1;
    sil := sil * i;
  END LOOP;
  RETURN sil;
END;
$$                -- zamknięcie bloku programowego
LANGUAGE plpgsql;    -- deklaracja języka

SELECT lab04.silnia(6);

DROP FUNCTION lab04.silnia(int);

-- Funkcja zwracająca nazwisko uczestnika o zadanym id. Typ zmiennej przypisany do typu kolumny.

CREATE OR REPLACE FUNCTION lab04.getnazwisko ( int ) RETURNS text AS
$$
  DECLARE
    id ALIAS FOR $1;                -- przypisanie atrybutu do parametru
    name lab04.uczestnik.nazwisko%TYPE;       -- przypisanie typu kolumny do
zmiennej
  BEGIN
    SELECT INTO name nazwisko FROM lab04.uczestnik
    WHERE id_uczestnik = iducz ;
    RETURN name;
  END;
$$ LANGUAGE 'plpgsql';

DROP FUNCTION lab04.getnazwisko ( int );
```

```
-- Typ zmiennej przypisany do typu rekordu w tabeli.

CREATE OR REPLACE FUNCTION lab04.getperson ( iducz int )
RETURNS text AS
$$
  DECLARE
    id ALIAS FOR $1;
    person lab04.uczestnik%ROWTYPE;      -- przypisanie typu rekordu
  BEGIN
    SELECT * INTO person FROM lab04.uczestnik
    WHERE id_uczestnik = id ;
```

```
    RETURN person.nazwisko || ' ' || person.imie;
  END;
$$ LANGUAGE 'plpgsql';

SELECT lab04.getperson(4);
DROP FUNCTION lab04.getperson(int);
```

```
-- Typ zmiennej przypisany do typu rekordu w tabeli.
CREATE OR REPLACE FUNCTION fun2a ( int )
RETURNS text AS
$$
  DECLARE
    iducz ALIAS FOR $1;
    name uczestnik%ROWTYPE;            -- przypisanie typu atrybutu do typu rekordu
  BEGIN
    SELECT * INTO name FROM uczestnik
    WHERE id_uczestnik = iducz ;
    RETURN name.imie || ' ' || name.nazwisko;
  END;
$$ LANGUAGE 'plpgsql';

SELECT fun2a(1) ;
```

```
CREATE OR REPLACE FUNCTION lab04.getpersondata ( int )
RETURNS text AS
$$
  DECLARE
    id ALIAS FOR $1;
    person RECORD;     -- przypisanie typu RECORD
  BEGIN
    SELECT  imie, nazwisko INTO person FROM lab04.uczestnik
    WHERE id_uczestnik = id ;
    RETURN person.nazwisko || ' ' || person.imie ;
  END;
$$ LANGUAGE 'plpgsql';

SELECT lab04.getpersondata(5) ;
```

```
DO $$
BEGIN
  RAISE INFO 'information message %', now() ;
  RAISE LOG 'log message %', now();
  RAISE DEBUG 'debug message %', now();
  RAISE WARNING 'warning message %', now();
```

```
  RAISE NOTICE 'notice message %', now();
END $$;

DO $$
DECLARE
  test text := '[brak danych]' ;
BEGIN
  -- ... kod
  -- ...
  -- raport o błędzie i podpowiedź
  RAISE EXCEPTION 'UWAGA: %', test
 USING HINT = 'Sprawdź parametry';
END $$;

DO $$
BEGIN
  RAISE SQLSTATE '00200' ;
END $$;
```

```
CREATE OR REPLACE FUNCTION lab04.getpersonC (id int) RETURNS text AS $$
DECLARE
    rec lab04.uczestnik%ROWTYPE;
BEGIN
    SELECT INTO rec *  FROM lab04.uczestnik WHERE id_uczestnik = id;
    IF NOT FOUND THEN
      RAISE EXCEPTION 'Brak danych w bazie dla id % ', id;
    END IF;
    RETURN  rec.imie || ' ' || rec.nazwisko;
END;
$$
LANGUAGE 'plpgsql';

SELECT lab04.getpersonC(9999);
```

```
CREATE OR REPLACE FUNCTION lab04.getperson_a (p_pattern VARCHAR)
 RETURNS TABLE ( im VARCHAR, naz VARCHAR ) AS
-- zwracany typ danych typu tablica
$$
BEGIN
 RETURN QUERY
   SELECT imie, nazwisko FROM lab04.uczestnik WHERE lower(nazwisko) LIKE
lower(p_pattern) ;
END;
```

```
$$ LANGUAGE 'plpgsql';

SELECT * FROM lab04.getperson_a('k%');
```

```
CREATE OR REPLACE FUNCTION lab04.getperson_b (p_pattern VARCHAR)
 RETURNS TABLE ( im VARCHAR, naz VARCHAR ) AS
$$
DECLARE
   rec RECORD;
BEGIN
 FOR rec IN (SELECT imie, nazwisko FROM lab04.uczestnik
                WHERE UPPER(nazwisko) LIKE UPPER(p_pattern) )
 LOOP
      im  := rec.imie ;
      naz := rec.nazwisko;
      RETURN NEXT;
 END LOOP;
END;
$$  LANGUAGE 'plpgsql';

SELECT * FROM lab04.getperson_b('k%');
```

```
CREATE OR REPLACE FUNCTION lab04.getperson_c ( sort_type char(1), n
INTEGER )
RETURNS TABLE ( im VARCHAR, naz VARCHAR, opis VARCHAR ) AS
$$
DECLARE
   rec RECORD;
   query text;
BEGIN
   query := 'SELECT u.imie, u.nazwisko, ko.opis FROM lab04.uczestnik u JOIN
lab04.uczest_kurs uk ON u.id_uczestnik=uk.id_uczest
                                        JOIN lab04.kurs k USING ( id_kurs )
                                        JOIN lab04.kurs_opis ko ON k.id_nazwa =
ko.id_kurs ';
  IF sort_type = 'U' THEN
    query := query || 'ORDER BY u.nazwisko, ko.opis ';
  ELSIF sort_type = 'G' THEN
    query := query || 'ORDER BY ko.opis, u.nazwisko ';
  ELSE
    RAISE EXCEPTION 'Niepoprawny typ sortowania %s', sort_type;
  END IF;
```

```
  query := query || ' LIMIT $1';

  FOR rec IN EXECUTE query USING n
    LOOP
    -- RAISE NOTICE '% - %', rec.release_year, rec.title;
      im   := rec.imie   ;
      naz  := rec.nazwisko  ;
      opis := rec.opis ;
      RETURN NEXT;
  END LOOP;

END;
$$ LANGUAGE plpgsql;

SELECT * FROM lab04.getperson_c('U', 2);
DROP FUNCTION lab04.getperson_c;
```

Funkcja z użyciem konstrukcji kursora.

```
CREATE OR REPLACE FUNCTION lab04.kursor (stext text) RETURNS text AS
$lab$
DECLARE
  records TEXT DEFAULT '';
  rec_wykladowca   RECORD;
  cur_wykladowcy CURSOR FOR SELECT * FROM lab04.wykladowca ;
  id INTEGER;
BEGIN
   id := 0 ;
   OPEN cur_wykladowcy ;          -- otwarcie kursora
   LOOP
    -- pobranie rekordu z kursora do zmiennej rec_wykladowc
      FETCH cur_wykladowcy INTO rec_wykladowca;
      EXIT WHEN NOT FOUND;        -- zamkniecie jak brak dalszych rekordow
    -- tworzenie rekordu wynikowego
      IF UPPER(rec_wykladowca.nazwisko) LIKE UPPER(stext) AND id != 0 THEN
        records := records || ',' || rec_wykladowca.nazwisko || ':' || rec_wykladowca.imie ||
COALESCE(':' || rec_wykladowca.email, '');
      END IF;
      IF UPPER(rec_wykladowca.nazwisko) LIKE UPPER(stext) AND id = 0 THEN
        records := rec_wykladowca.nazwisko || ':' || rec_wykladowca.imie ||
COALESCE(':' || rec_wykladowca.email, '');
        id := 1 ;
      END IF;

   END LOOP;
   CLOSE cur_wykladowcy;          -- zamkniecie kursora
   RETURN records;
END;
$lab$ LANGUAGE plpgsql;

SELECT lab04.kursor( 'K%');
```