zobrazení délky ve výpisu v psql

```
postgres=# \dt+ codebooks.*
                        List of relations
  Schema    |    Name    | Type  | Owner |    Size    | Description
------------+------------+-------+-------+------------+-------------
 codebooks | lau1       | table | pavel | 8192 bytes |
 codebooks | lau2       | table | pavel | 448 kB     |
 codebooks | nuts0      | table | pavel | 8192 bytes |
 codebooks | nuts1      | table | pavel | 8192 bytes |
 codebooks | nuts2      | table | pavel | 8192 bytes |
 codebooks | nuts3      | table | pavel | 8192 bytes |
 codebooks | psc_obce   | table | pavel | 1352 kB    |
(7 rows)
```

Lámání textu ve sloupcích

```
postgres=# \pset format wrapped
Output format is wrapped.

postgres=# select 'příliš žlutý kůň se napil žluté vody', 'příliš žlutý kůň
se napil žluté vody';
             ?column?              |             ?column?
-----------------------------------+----------------------------------
 příliš žlutý kůň se napil žluté  | příliš žlutý kůň se napil žluté v
 vody                              ; ody
(1 row)
```

PL/pgSQL -EXECUTE USING

- Ochrana před SQL injektáží
- Optimální provádění SQL dotazu
  - náhrada parametrů konstantami
- Dynamické získání položky recordu

Ochrana před SQL injektáží

```
EXECUTE 'SELECT * FROM users WHERE
 username = $1'
 USING _users;
```

Optimální provádění dotazu

```
CREATE OR REPLACE FUNCTION show_obce(_obec varchar)
  RETURNS SETOF varchar AS $$
  DECLARE s varchar;
  BEGIN
    /* pokud je obec nezadaná, ignoruj ji */
    s := 'SELECT * FROM codebooks.psc_obce';
    IF NOT s IS NULL THEN
      s := s || ' WHERE obec = ' || quote_literal(_obec);
    END IF;
    RETURN QUERY EXECUTE s;
    RETURN;
  END;
  $$ LANGUAGE plpgsql;
```

Optimální provádění dotazu

```
CREATE OR REPLACE FUNCTION show_obce(_obec varchar)
 RETURNS SETOF varchar AS $$
 DECLARE s varchar;
 BEGIN
   FOR s IN EXPLAIN
               SELECT *
                  FROM codebooks.psc_obce
                 WHERE _obec IS NULL OR obec = _obec
   LOOP
     RAISE NOTICE '%', s;
   END LOOP;
   RETURN QUERY SELECT obec
                     FROM codebooks.psc_obce
                    WHERE _obec IS NULL OR obec = _obec;
   RETURN;
 END;
 $$ LANGUAGE plpgsql;
```

Optimální provádění dotazu

```
postgres=# select * from show_obce(NULL) limit 3;
NOTICE:  Seq Scan on psc_obce  (cost=0.00..376.98 rows=83 width=51)
NOTICE:    Filter: (($1 IS NULL) OR ((obec)::text = ($1)::text))
 show_obce
-----------
 Větřní
 Malečov
 Podlesí
(3 rows)

Time: 16,721 ms
postgres=# select * from show_obce('Benešov' limit 3;
NOTICE:  Seq Scan on psc_obce  (cost=0.00..376.98 rows=88 width=51)
NOTICE:    Filter: (($1 IS NULL) OR ((obec)::text = ($1)::text))
 show_obce
-----------
 Benešov
 Benešov
 Benešov
(3 rows)

Time: 13,408 ms
```

Optimální provádění dotazu

```
CREATE OR REPLACE FUNCTION show_obce(_obec varchar)
 RETURNS SETOF varchar AS $$
 DECLARE s varchar;
 BEGIN
   FOR s IN EXECUTE 'EXPLAIN
               SELECT *
                  FROM codebooks.psc_obce
                 WHERE $1 IS NULL OR obec = $1'
            USING _obec
   LOOP
     RAISE NOTICE '%', s;
   END LOOP;
   RETURN QUERY EXECUTE 'SELECT obec
                    FROM codebooks.psc_obce
                   WHERE $1 IS NULL OR obec = $1'
                 USING _obec;
   RETURN;
 END;
 $$ LANGUAGE plpgsql;
```

Optimální provádění dotazu

```
postgres=# select * from show_obce(NULL) limit 3;
NOTICE:  Seq Scan on psc_obce  (cost=0.00..335.38 rows=16638 width=51)
 show_obce
-----------
 Větřní
 Malečov
 Podlesí
(3 rows)

Time: 18,933 ms
postgres=# select * from show_obce('Benešov') limit 3;
NOTICE:  Bitmap Heap Scan on psc_obce  (cost=4.29..21.77 rows=5 width=51)
NOTICE:    Recheck Cond: ((obec)::text = 'Benešov'::text)
NOTICE:    ->  Bitmap Index Scan on obec_idx  (cost=0.00..4.29 rows=5
  width=0)
NOTICE:          Index Cond: ((obec)::text = 'Benešov'::text)
 show_obce
-----------
 Benešov
 Benešov
 Benešov
(3 rows)

Time: 2,704 ms
```

Dynamické získání položky recordu

```
CREATE OR REPLACE FUNCTION foot()
RETURNS TRIGGER AS $$
DECLARE
 t text;
begin
  RAISE NOTICE '%', get_field(NEW, 'a');
  RAISE NOTICE '%', get_field(NEW, 'b');
  RETURN new;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get_field(rec anyelement, field varchar)
RETURNS TEXT AS $$
DECLARE s varchar;
BEGIN
  EXECUTE 'SELECT $1' || '.' || field || '::text'
    INTO s
    USING rec;
  RETURN s;
END;
$$ LANGUAGE plpgsql;
```

Fulltextové vyhledávání prefixu

```
postgres=# create index full_idx
             on codebooks.psc_obce using gin ((to_tsvector('simple', cast_obce)));
CREATE INDEX
Time: 214,273 ms

postgres=# select * from codebooks.psc_obce
             where to_tsvector('simple', cast_obce) @@ to_tsquery('simple', 'Benešov');
         obec           |         cast_obce          |  psc  |      nazev_posty       |  lau1
------------------------+----------------------------+-------+------------------------+-------
 Broumov                | Benešov                    | 55001 | Broumov 1              | CZ0523
 Benešov                | Benešov                    | 25601 | Benešov u Prahy        | CZ0201
 Benešov                | Benešov                    | 67953 | Benešov u Boskovic     | CZ0641
 Černovice              | Benešov                    | 39470 | Kamenice nad Lipou     | CZ0633
 Benešov nad Černou     | Benešov nad Černou (část)  | 38241 | Kaplice 1              | CZ0312
 Benešov nad Černou     | Benešov nad Černou (část)  | 38282 | Benešov nad Černou     | CZ0312
 Benešov nad Ploučnicí  | Benešov nad Ploučnicí      | 40722 | Benešov nad Ploučnicí  | CZ0421
 Benešov u Semil        | Benešov u Semil            | 51206 | Benešov u Semil        | CZ0514
 Dolní Benešov          | Dolní Benešov              | 74722 | Dolní Benešov          | CZ0805
 Horní Benešov          | Horní Benešov              | 79312 | Horní Benešov          | CZ0801
(10 rows)

Time: 101,172 ms
```

Fulltextové vyhledávání prefixu

```
-- vyžaduje GIN index
postgres=# select *
             from codebooks.psc_obce
            where to_tsvector('simple', cast_obce) @@ to_tsquery('simple', 'Bene:*');
         obec           |         cast_obce          |  psc  |      nazev_posty      |  lau1
------------------------+----------------------------+-------+-----------------------+-------
 Benecko                | Benecko (část)             | 51237 | Benecko               | CZ0514
 Benecko                | Benecko (část)             | 51401 | Jilemnice             | CZ0514
 Bušanovice             | Beneda                     | 38422 | Vlachovo Březí        | CZ0315
 Broumov                | Benešov                    | 55001 | Broumov 1             | CZ0523
 Benešov                | Benešov                    | 25601 | Benešov u Prahy       | CZ0201
 Benešov                | Benešov                    | 67953 | Benešov u Boskovic    | CZ0641
 Černovice              | Benešov                    | 39470 | Kamenice nad Lipou    | CZ0633
 Benešov nad Černou     | Benešov nad Černou (část)  | 38241 | Kaplice 1             | CZ0312
 Benešov nad Černou     | Benešov nad Černou (část)  | 38282 | Benešov nad Černou    | CZ0312
 Benešov nad Ploučnicí  | Benešov nad Ploučnicí      | 40722 | Benešov nad Ploučnicí | CZ0421
  ....
 Všelibice              | Benešovice                 | 46348 | Všelibice             | CZ0513
 Benetice               | Benetice                   | 67506 | Benetice              | CZ0634
 Světlá nad Sázavou     | Benetice                   | 58291 | Světlá nad Sázavou    | CZ0631
 Dolní Benešov          | Dolní Benešov              | 74722 | Dolní Benešov         | CZ0805
 Horní Benešov          | Horní Benešov              | 79312 | Horní Benešov         | CZ0801
(19 rows)

Time: 3,076 ms
```

Fulltextové vyhledávání prefixu

```
postgres=# select * from codebooks.psc_obce where cast_obce ilike '%Bene%';
         obec           |          cast_obce          |  psc  |             nazev_posty             |  lau1
------------------------+-----------------------------+-------+-------------------------------------+--------
 Benecko                | Benecko (část)              | 51237 | Benecko                             | CZ0514
 Benecko                | Benecko (část)              | 51401 | Jilemnice                           | CZ0514
 Bušanovice             | Beneda                      | 38422 | Vlachovo Březí                      | CZ0315
 Broumov                | Benešov                     | 55001 | Broumov 1                           | CZ0523
 Benešov                | Benešov                     | 25601 | Benešov u Prahy                     | CZ0201
 Benešov                | Benešov                     | 67953 | Benešov u Boskovic                  | CZ0641
 Černovice              | Benešov                     | 39470 | Kamenice nad Lipou                  | CZ0633
 Benešov nad Černou     | Benešov nad Černou (část)   | 38241 | Kaplice 1                           | CZ0312
 Benešov nad Černou     | Benešov nad Černou (část)   | 38282 | Benešov nad Černou                  | CZ0312
 Benešov nad Ploučnicí  | Benešov nad Ploučnicí       | 40722 | Benešov nad Ploučnicí               | CZ0421
   ....
 Dívčice                | Dubenec                     | 37348 | Dívčice                             | CZ0311
 Dubenec                | Dubenec                     | 54455 | Dubenec u Dvora Králové nad Labem   | CZ0525
 Horní Benešov          | Horní Benešov               | 79312 | Horní Benešov                       | CZ0801
 Křivsoudov             | Lhota Bubeneč               | 25765 | Čechtice                            | CZ0201
 Lubenec                | Lubenec                     | 43983 | Lubenec                             | CZ0424
 Praha                  | Bubeneč (Praha 6)           | 16000 | Praha 6                             | CZ0100
 Praha                  | Bubeneč (Praha 7)           | 17000 | Praha 7                             | CZ0100
(26 rows)

Time: 59,856 ms
```

Funkce generate_subscripts

```
create or replace function unnest(anyarray)
returns setof anyelement as $$
  select $1[i]
    from generate_subscripts($1,1) g(i);
$$ language sql immutable strict;

postgres=# select * from unnest(array[10,20,30]);
 unnest
--------
     10
     20
     30
(3 rows)
```

Lokalizované výstupy fce to_char

```
postgres=# select to_char(current_date,
                          'tmDay, DD. tmMonth');
      to_char
--------------------
 Středa, 11. Červen
(1 row)
```

Rozšíření zachycení výjimek

```
WHEN division_by_zero THEN ...
WHEN SQLSTATE '22012' THEN ...
```

Vlastní výjimky

```
RAISE division_by_zero;
RAISE SQLSTATE '22012';

RAISE EXCEPTION 'Nonexistent ID --> %', user_id
    USING HINT = 'Please check your user id';

RAISE 'Duplicate user ID: %', user_id
    USING ERRCODE = 'unique_violation';
RAISE 'Duplicate user ID: %', user_id
    USING ERRCODE = '23505';
```

Parametrizované pohledy
(inline SRF funkcí)

```
postgres=# create or replace function filter(varchar)
           returns setof codebooks.psc_obce as $$
           select *
              from codebooks.psc_obce
             where lower(obec) = lower($1);
           $$ language sql stable;
CREATE FUNCTION
Time: 3,453 ms

postgres=# explain select * from filter('Benešov');
                              QUERY PLAN
-----------------------------------------------------------------------
 Bitmap Heap Scan on psc_obce  (cost=4.29..21.79 rows=5 width=50)
   Recheck Cond: (lower((obec)::text) = 'benešov'::text)
   ->  Bitmap Index Scan on fx  (cost=0.00..4.29 rows=5 width=0)
         Index Cond: (lower((obec)::text) = 'benešov'::text)
(4 rows)

Time: 4,329 ms
```

Parametrizované pohledy

```
postgres=# create or replace function filter(varchar)
           returns setof codebooks.psc_obce as $$
           select *
              from codebooks.psc_obce
             where lower(obec) = lower($1);
           $$ language sql volatile;
CREATE FUNCTION
Time: 3,453 ms

postgres=# explain select * from filter('Benešov');
                            QUERY PLAN
-------------------------------------------------------------------
 Function Scan on filter  (cost=0.00..260.00 rows=1000 width=418)
(1 row)

Time: 1,326 ms
```

PL/pgSQL konstrukce CASE

```
/* simple case */
CASE x
     WHEN 1, 2 THEN
          msg := 'one or two';
     ELSE
          msg := 'other value than one or two';
END CASE;

/* search case */
CASE
     WHEN x BETWEEN 0 AND 10 THEN
          msg := 'value is between zero and ten';
     WHEN x BETWEEN 11 AND 20 THEN
          msg := 'value is between eleven and twenty';
END CASE;
```

TRUNCATE trigger

```
CREATE TRIGGER trgtr
    BEFORE TRUNCATE ON foo
   FOR EACH STATEMENT
    EXECUTE PROCEDURE trgfce();
```

Hash pro DISTINCT, UNION ...

```
postgres=# explain select distinct * from foo;
                          QUERY PLAN
-------------------------------------------------------------
 HashAggregate  (cost=165.00..174.76 rows=976 width=4)
   ->  Seq Scan on foo  (cost=0.00..140.00 rows=10000 width=4)
(2 rows)
```

```
postgres=# explain select * from foo union select * from foo union
  select * from foo;
                             QUERY PLAN
---------------------------------------------------------------------
 HashAggregate  (cost=795.00..1095.00 rows=30000 width=4)
   ->  Append  (cost=0.00..720.00 rows=30000 width=4)
         ->  Seq Scan on foo  (cost=0.00..140.00 rows=10000 width=4)
         ->  Seq Scan on foo  (cost=0.00..140.00 rows=10000 width=4)
         ->  Seq Scan on foo  (cost=0.00..140.00 rows=10000 width=4)
(5 rows)
```

CASE INSENSITIVE Text type

```
postgres=# \i /usr/local/pgsql/share/contrib/citext.sql
postgres=# create table testci(email citext);
CREATE TABLE
Time: 9,377 ms
postgres=# insert into testci
  values('Pavel.Stehule@Gmail.CZ');
INSERT 0 1
postgres=# select * from testci where email like '%gmail.cz';
          email
------------------------
 Pavel.Stehule@Gmail.CZ
(1 row)
postgres=# select * from testci
               where email = 'pavel.stehule@gmail.cz';
          email
------------------------
 Pavel.Stehule@Gmail.CZ
(1 row)
```

Variadické funkce

```
CREATE FUNCTION myleast(VARIADIC a numeric[])
  RETURNS NUMERIC AS $$
    SELECT min($1[i])
       FROM generate_subscripts($1,1) g(i)
  $$ LANGUAGE SQL IMMUTABLE;

CREATE FUNCTION
Time: 5,274 ms
postgres=# SELECT myleast(1,2,3,4);
 myleast
---------
       1
(1 row)
```

Variadické funkce

```
postgres=# SELECT myleast(VARIADIC ARRAY[1,3,4,-5,6,8]);
 myleast
---------
      -5
(1 row)
```

Tabulkové funkce

```
CREATE OR REPLACE FUNCTION fib(int)
  RETURNS TABLE(v int) AS $$
  DECLARE s1 int := 1;
          s0 int := 0;
  BEGIN v := 0;
    WHILE v < $1
    LOOP
      RETURN NEXT;
      s0 := s1 + v;
      v := s1; s1 := s0;
    END LOOP;
    RETURN;
  END;
  $$ LANGUAGE plpgsql;
```

SEMIJOIN pro EXISTS

```
postgres=# explain select * from film where film_id in (select film_id from
  film_actor );
                                  QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=117.26..195.78 rows=977 width=390)
   Hash Cond: (film.film_id = film_actor.film_id)
   ->  Seq Scan on film  (cost=0.00..65.00 rows=1000 width=390)
   ->  Hash  (cost=105.05..105.05 rows=977 width=2)
         ->  HashAggregate  (cost=95.28..105.05 rows=977 width=2)
               ->  Seq Scan on film_actor  (cost=0.00..81.62 rows=5462 width=2)
(6 rows)

postgres=# explain select * from film f where exists (select * from film_actor
  where film_id = f.film_id);
                                  QUERY PLAN
------------------------------------------------------------------------------
 Hash Join  (cost=117.26..195.78 rows=977 width=390)
   Hash Cond: (f.film_id = film_actor.film_id)
   ->  Seq Scan on film f  (cost=0.00..65.00 rows=1000 width=390)
   ->  Hash  (cost=105.05..105.05 rows=977 width=2)
         ->  HashAggregate  (cost=95.28..105.05 rows=977 width=2)
               ->  Seq Scan on film_actor  (cost=0.00..81.62 rows=5462 width=2)
(6 rows)
```

ANTIJOIN pro EXISTS

```
postgres=# explain select * from film f where not exists (select * from
  film_actor where film_id = f.film_id);
                                QUERY PLAN
-------------------------------------------------------------------------
 Hash Anti Join  (cost=149.90..245.12 rows=23 width=390)
   Hash Cond: (f.film_id = film_actor.film_id)
   ->  Seq Scan on film f  (cost=0.00..65.00 rows=1000 width=390)
   ->  Hash  (cost=81.62..81.62 rows=5462 width=2)
         ->  Seq Scan on film_actor  (cost=0.00..81.62 rows=5462 width=2)
(5 rows)
```

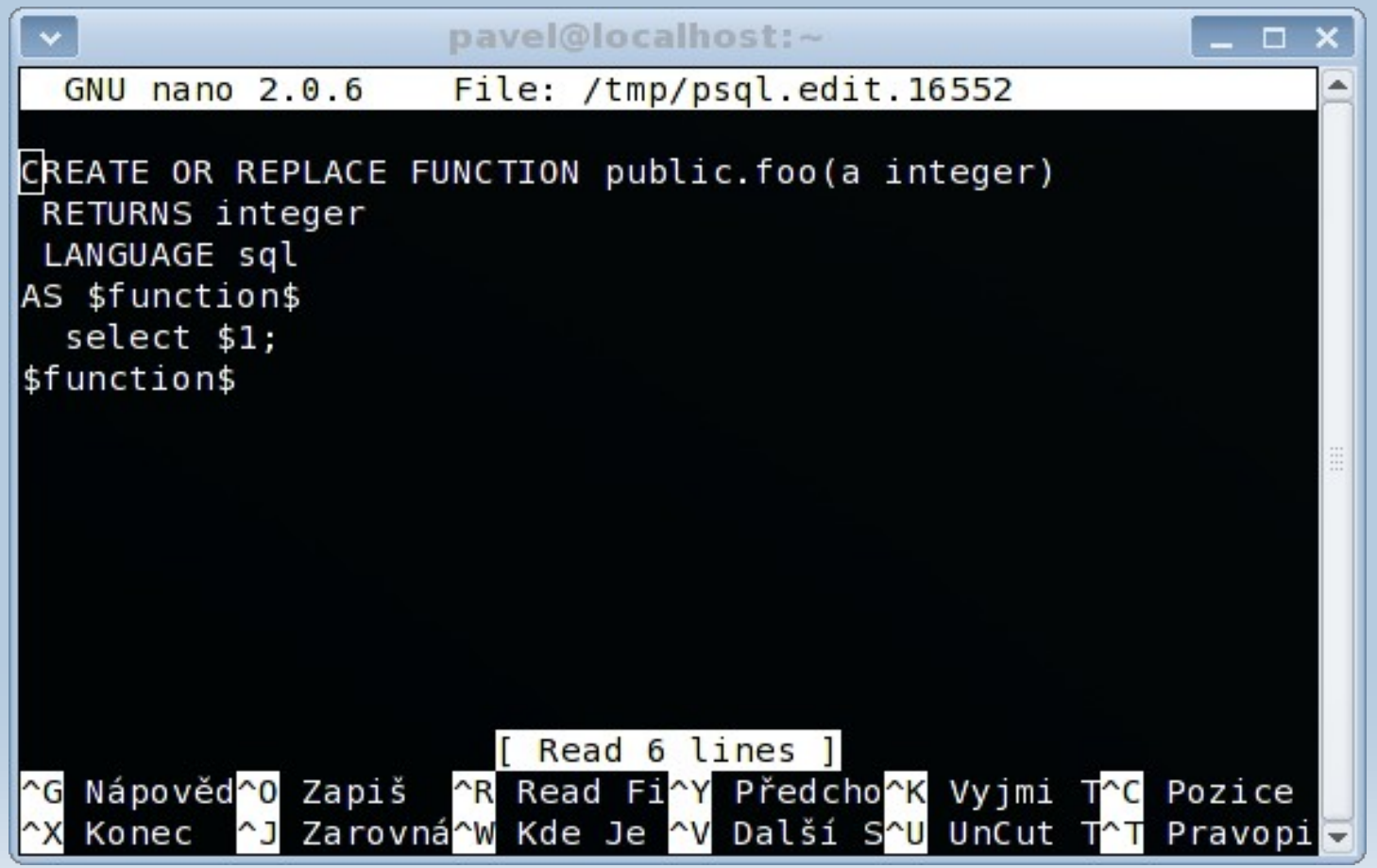
Spuštění externího editoru pro editaci funkcí

```
[pavel@localhost ~]$ psql postgres
psql (8.4devel)
Type "help" for help.

postgres=# create function foo(a int)
postgres-# returns int as $$
postgres$#   select $1;
postgres$# $$ language sql;
CREATE FUNCTION
Time: 4,510 ms
postgres=# select foo(10);
 foo
-----
  10
(1 row)

Time: 0,735 ms
postgres=#
```

Spuštění externího editoru pro editaci funkcí

```
[pavel@localhost ~]$ export EDITOR=nano
[pavel@localhost ~]$ psql postgres
psql (8.4devel)
Type "help" for help.
postgres=# \ef foo
postgres-# ;
CREATE FUNCTION
Time: 3,605 ms
```

Spuštění externího editoru pro editaci funkcí



Separátní locales pro každou databázi

```
[pavel@localhost pgsql]$ /usr/local/pgsql/bin/psql -l
                                    List of databases
   Name    |  Owner   | Encoding |    Collation    |      Ctype      |          Access Privileges
-----------+----------+----------+-----------------+-----------------+-------------------------------------
 postgres  | postgres | UTF8     | cs_CZ.UTF-8     | cs_CZ.UTF-8     |
 template0 | postgres | UTF8     | cs_CZ.UTF-8     | cs_CZ.UTF-8     | {=c/postgres,postgres=CTc/postgres}
 template1 | postgres | UTF8     | cs_CZ.UTF-8     | cs_CZ.UTF-8     | {=c/postgres,postgres=CTc/postgres}
 testdb    | pavel    | LATIN2   | cs_CZ.iso8859-2 | cs_CZ.iso8859-2 |
(4 rows)

testdb=# set client_encoding to utf8;
SET
Time: 1,133 ms
testdb=# select upper('žšššžšž');
 upper
--------
 ŽŠŠŽŠŽ
(1 row)

testdb=# select octet_length('žýř');
 octet_length
--------------
            3
(1 row)
```

Separátní locales pro každou databázi

```
pavel@localhost pgsql]$ psql postgres
psql (8.4devel)
Type "help" for help.

postgres=# select octet_length('žýř');
 octet_length
--------------
            6
(1 row)

Time: 1,516 ms
postgres=#
```

Statistiky pro fulltext

```
postgres=# explain analyze
               select * from film where fulltext @@ to_tsquery('dog');
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Bitmap Heap Scan on film  (cost=5.02..62.42 rows=99 width=390)
 (actual time=0.509..1.854 rows=99 loops=1)
   Recheck Cond: (fulltext @@ to_tsquery('dog'::text))
   ->  Bitmap Index Scan on film_fulltext_idx
       (cost=0.00..5.00 rows=99 width=0)
       (actual time=0.456..0.456 rows=99 loops=1)
         Index Cond: (fulltext @@ to_tsquery('dog'::text))
 Total runtime: 2.230 ms
(5 rows)
```

HASH index

```
postgres=# create index hash_idx_title on film
             using hash(title);
CREATE INDEX
postgres=# explain select title from film where title = 'DRAGON SQUAD';
                                 QUERY PLAN
-----------------------------------------------------------------------------
 Index Scan using hash_idx_title on film  (cost=0.00..8.27 rows=1 width=15)
   Index Cond: ((title)::text = 'DRAGON SQUAD'::text)
(2 rows)
```

Common Table Expression (nerekurzivní)

```
postgres=# select * from prodej;
      nazev       |    kategorie     | cena
------------------+------------------+-------
 jogurt           | mléčné produkty | 20.00
 chléb - šumava   | pečivo           | 10.00
 plnotučné mléko | mléčné produkty | 15.00
 rohlíky          | pečivo           | 12.00
(4 rows)
```

Common Table Expression
(nerekurzivní – celkový součet)

```
postgres=# with prehled as
               (select kategorie, sum(cena) cena
                   from prodej
                  group by kategorie
               ) select *
                   from prehled
                 union all
                 select 'Celkem', sum(cena)
                   from prehled;
    kategorie     | cena
------------------+-------
 pečivo           | 22.00
 mléčné produkty | 35.00
 Celkem           | 57.00
(3 rows)
```

Common Table Expression (rekurzivní)

```
postgres=# select * from org_schema;
 jmeno   | nadrizeny
---------+-----------
 Libor   |  NULL
 Vráťa   | Libor
 Petra   | Libor
 Martin  | Vráťa
 Pavel   | Martin
 Marek   | Martin
 Robin   | Petra
(7 rows)
```

Common Table Expression (rekurzivní)

```
postgres=# with recursive rq as
             (select 0 as level, jmeno as path, *
                 from org_schema where nadrizeny is null
              union all
              select level+1 as lev,
                     path || e'\\' || o.jmeno as path, o.*
                 from rq, org_schema o
                where rq.jmeno = o.nadrizeny
             ) select *, repeat('  ', level) || jmeno
                 from rq order by path;
 lev |           path            | jmeno  | nadrizeny |  ?column?
-----+---------------------------+--------+-----------+------------
   0 | Libor                     | Libor  |  NULL     | Libor
   1 | Libor\Petra               | Petra  | Libor     |   Petra
   2 | Libor\Petra\Robin         | Robin  | Petra     |     Robin
   1 | Libor\Vráťa               | Vráťa  | Libor     |   Vráťa
   2 | Libor\Vráťa\Martin        | Martin | Vráťa     |     Martin
   3 | Libor\Vráťa\Martin\Marek  | Marek  | Martin    |       Marek
   3 | Libor\Vráťa\Martin\Pavel  | Pavel  | Martin    |       Pavel
(7 rows)
```

Infinity::date

```
postgres=# select 'infinity' >
                  current_date + interval '10000 years';
 ?column?
----------
 t
(1 row)

postgres=# select '-infinity' <
                  current_date - interval '6000 years';
 ?column?
----------
 t
(1 row)
```

ANSI SQL 2008 ekvivalent klauzule LIMIT

```
postgres=# select *
                from generate_series(1,100)
               offset 10 rows
               fetch next 5 rows only;
 generate_series
-----------------
              11
              12
              13
              14
              15
(5 rows)
postgres=# select * from generate_series(1,100)
               limit 5
               offset 10;
 generate_series
-----------------
              11
          ...
(5 rows)
```

Podpora Unicode konstant

```
postgres=# SELECT U&'euro - \20AC, Ř -  \0158, ju - \044e';
          ?column?
---------------------------
 euro - €, Ř -  Ř, ju - ю
(1 row)

postgres=# SELECT U&'řečtina \03ec *03e3 *03ea' uescape '*';
     ?column?
-------------------
 řečtina \03ec ϣ Ϫ
(1 row)
```

IO Cast

```
CREATE CAST (int4 AS casttesttype) WITH INOUT;
SELECT 1234::int4::casttesttype; -- Should work now
 casttesttype
--------------
 1234
(1 row)
```

Podpora klauzule RETURNING v SQL funkcích

```
CREATE FUNCTION fx()
RETURNS SETOF integer AS $$
  INSERT INTO foo VALUES(DEFAULT)
    RETURNING *
$$ LANGUAGE sql VOLATILE;

postgres=# SELECT * FROM fx() AS f(a)
              WHERE a > 10;
 a
----
 12
(1 row)
```