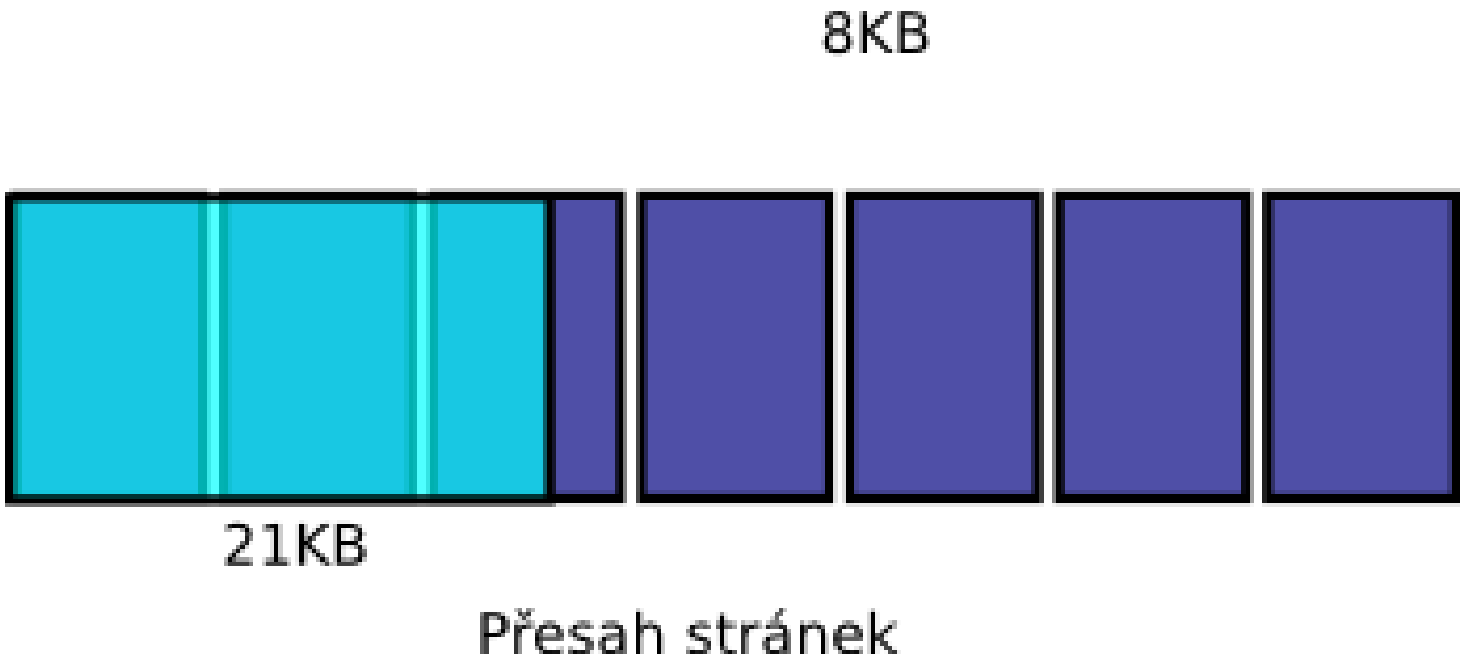
PostgreSQL

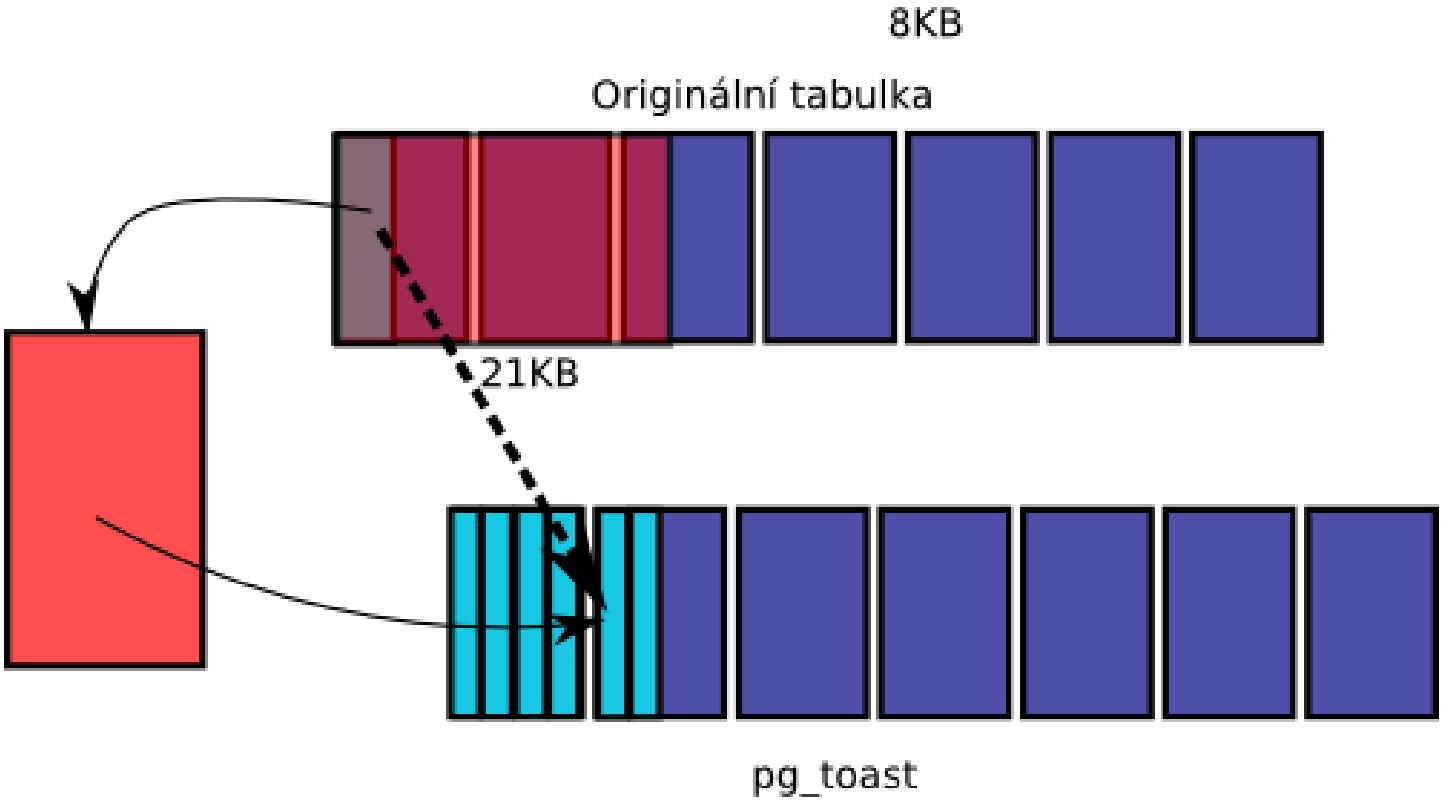
- Vzniká jako akademický projekt
  - Experimentální vlastnosti
    - Podpora dědičnosti
    - Rozšiřitelnost – vlastní datové typy
  - Univerzální nasazení ve vědecké sféře
    - Obsahuje podporu polí (časové řady)
    - Geotypy – bod, polygon

TOAST

- Způsob uložení větších až velkých dat v PostgreSQL
  - Limitem, který je třeba překonat je velikost datové stránky
    - Zvětšením datové stránky
    - Možností přesahu hranice datové stránky
    - TOAST

TOAST

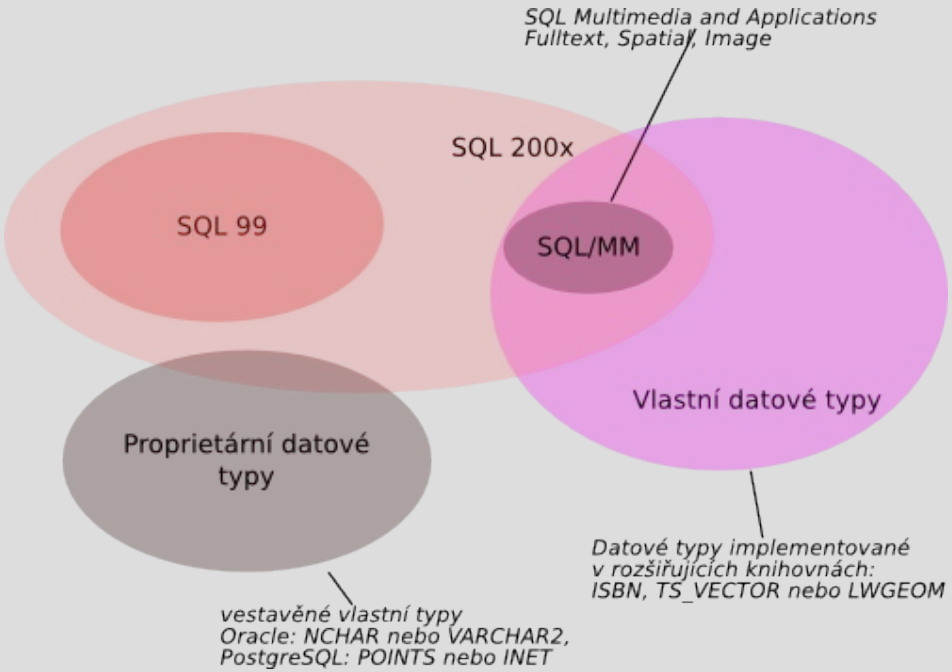



8KB
Originální tabulka
21KB
pg_toast

TOAST

- Originální data jsou komprimována a rozdělené do „kousků“
- Tyto kousky jsou uložené do tabulky pg_toast
- Místo vlastních dat je uložen ukazatel do tabulky pg_toast
- Toast se aktivuje, pokud záznam před toustováním je delší než 2KB
- Teoretický limit 1GB, praktický 10-20MB (záleží na velikosti operační paměti).

Datové typy v SQL



Dědičnost tabulek

```
postgres=# create table mesta(nazev varchar);
CREATE TABLE
postgres=# create table mesta_cs (nazev varchar) inherits (mesta);
CREATE TABLE
postgres=# create table mesta_en (nazev varchar) inherits (mesta);
CREATE TABLE
postgres=# insert into mesta_en values('Londyn');
INSERT 0 1
postgres=# insert into mesta_cs values('Praha');
INSERT 0 1
postgres=# select * from mesta_cs;
 nazev
-------
 Praha
(1 row)

postgres=# select * from mesta;
 nazev
--------
 Praha
 Londyn
(2 rows)
```

Domény - definice

```
postgres=# create domain cistr as varchar;
CREATE DOMAIN
postgres=# select cistr 'fooo';
 cistr
-------
 fooo
(1 row)

postgres=# select cistr 'Fooo';
 cistr
-------
 Fooo
(1 row)
```

Domény – vlastní funkce

```
create function cistreq(cistr, cistr)
returns boolean as $$
  select lower(trim($1)) = lower(trim($2));
$$ language sql immutable strict;

postgres=# select cistreq('a', ' A');
 cistreq
---------
 t
(1 row)

postgres=# select cistreq('a', ' A'::cistr);
 cistreq
---------
 t
(1 row)
```

Domény – vlastní operátory

```
postgres=# create operator = (
                    procedure = cistreq,
                    leftarg = cistr, rightarg = cistr);
CREATE OPERATOR

postgres=# select cistr 'Ahoj' = 'AHOJ';
 ?column?
----------
 t
(1 row)
```

Domény - test

```
postgres=# create table test(c cistr);
CREATE TABLE

postgres=# insert into test values(' ahoj '), ('Ahoj'), ('AHOJ');
INSERT 0 3

postgres=# select * from test;
   c
--------
  ahoj
 Ahoj
 AHOJ
(3 rows)
```

Domény - test

```
postgres=# select * from test where c = 'ahoj';
   c
--------
  ahoj
 Ahoj
 AHOJ
(3 rows)

postgres=# select * from test where c::varchar =
'ahoj';
 c
---
(0 rows)
```

Domény - test

```
postgres=# create index fxidx on test(lower(trim(c)));
CREATE INDEX

postgres=# explain select * from test where c = 'ahoj';
                                        QUERY PLAN
-------------------------------------------------------------------------------------
 Index Scan using fxidx on test  (cost=0.01..8.28 rows=1 width=32)
   Index Cond: (lower(btrim((c)::text)) = lower(btrim(('ahoj'::character varying)::text)))
(2 rows)
```

Domény - zhodnocení

- Vhodné pro základní návrh vlastních typů
- Nevhodné pro implementaci složitějších typů
  - Chybí vstupní/výstupní funkce
  - Přetypování vychází z bázických typů (nebere v potaz domény)

Vlastní binární typy

Mezi vlastními binárními typy a vestavěnými je jediný rozdíl – umístění knihovny – vše ostatní je stejné. Výkonostně, chováním se vlastní binární typy neliší od vestavěných.

Vsuvka – datové typy PostgreSQL

- Fixní – cokoliv s pevnou délkou do 8byte
  - Int, time, date, timestamp, interval, double
- Variabilní tzv. **varlena**
  - Všechno ostatní – text, varchar, numeric, bytea
    - Kratké do 255 byte (délka je v 1byte)
    - Dlouhé – ostatní (délka je ve 4 byte)
  - Připomíná string TurboPascalu – prvních n byte nese délku – rozdíl není to efektivní délka, nýbrž celková délka

Typ **Datum**

- Interní datový typ používaný v PostgreSQL pro přenos paramtrů SQL funkcí
  - Union
    - 8byte hodnoty fixního typu
    - 8byte pointer na typ varlena

Vstupně/výstupní funkce

```
Datum
cistrin(PG_FUNCTION_ARGS)
{
        Datum   str = textin(fcinfo);

        return DirectFunctionCall1(lower,
                           DirectFunctionCall1(btrim1, str));


}

Datum
cistrout(PG_FUNCTION_ARGS)
{
        return textout(fcinfo);
}
```

Registrace funkcí pro SQL

```
CREATE TYPE cistr;

CREATE OR REPLACE FUNCTION cistrin(cstring)
RETURNS cistr
AS '$libdir/cistr'
LANGUAGE C IMMUTABLE STRICT;

CREATE OR REPLACE FUNCTION cistrout(cistr)
RETURNS cstring
AS '$libdir/cistr'
LANGUAGE C IMMUTABLE STRICT;
```

Registrace typu

```
CREATE TYPE cistr (
    INPUT          = cistrin,
    OUTPUT         = cistrout,
    INTERNALLENGTH = VARIABLE,
    STORAGE        = extended,
    CATEGORY       = 'S',
    PREFERRED      = false
);
```

Test typu cistr

```
postgres=# select '>>>' || ' AHoj '::cistr || '<<<';
  ?column?
------------
 >>>ahoj<<<
(1 row)
```

Registrace operátoru

```
CREATE OPERATOR = (
    LEFTARG    = cistr,
    RIGHTARG   = cistr,
    COMMUTATOR = =,
    NEGATOR    = <>,
    PROCEDURE  = cistreq,
    RESTRICT   = eqsel,
    JOIN       = eqjoinsel,
    HASHES,
    MERGES
);
```

Test porovnání

```
postgres=# select ' Ahoj '::cistr = 'ahoj';
 ?column?
----------
 t
(1 row)
```

Závěrečný test

```
postgres=# create table test(c cistr);
CREATE TABLE
postgres=# insert into test values(' ahoj '),
('Ahoj'), ('AHOJ');
INSERT 0 3
postgres=# select * from test;
  c
------
 ahoj
 ahoj
 ahoj
(3 rows)
```

Závěrečný test

```
postgres=# select count(*) from test where c = ' Ahoj ';
 count
-------
     3
(1 row)

postgres=# select count(*) from test where c = 'AHOJ';
 count
-------
     3
(1 row)
```

Složené typy

```
postgres=# create type p as (x int, y int);
CREATE TYPE
postgres=# create table ptab(bod p);
CREATE TABLE
postgres=# insert into ptab values((10,20));
INSERT 0 1
postgres=# select * from ptab;
   bod
---------
 (10,20)
(1 row)
```

Složené typy

```
postgres=# select (bod).x, (bod).y from ptab;
 x  | y
----+----
 10 | 20
(1 row)

postgres=# select (bod).* from ptab;
 x  | y
----+----
 10 | 20
(1 row)
```

WKT a WKB formát

- Umožňuje zadání libovolných prostorových dat
- POINT(-126.4 45.32),
  MULTIPOINT((0 0), (0 1))
- Data WKT jsou uložena v textovém formátu
- Pokud databáze podporuje typ BYTEA mohou být transformovány do WKB formátu
- PostGIS provádí automaticky serializaci do WKB formátu

Vztah OpenGISu, PostGISu a SQL/MM