Školení PostgreSQL efektivně

Všeobecná část + administrace - podklady

Pavel Stěhule

http://www.postgres.cz

16. 10. 2023

Pavel Stěhule
pavel.stehule@gmail.com

- vývojář PostgreSQL od roku 1999 - vývoj PL/pgSQL,
- lektor PostgreSQL a SQL od roku 2006,
- instalace, migrace databází, audit aplikací postavených nad PostgreSQL,
- vývoj zákaznických modulů do PostgreSQL - více na http://www.postgres.cz/index.php/Služby,
- inhouse školení PostgreSQL,
- veřejná školení SQL a PLpgSQL,
- konzultace ohledně problémů s výkonem (performance) PostgreSQL,
- komerční podpora PostgreSQL

Informace a podpora PostgreSQL na internetu

- http://wiki.postgresql.org
- http://www.postgresql.org
- http://archives.postgresql.org
- http://www.postgres.cz
- http://groups.google.com/group/postgresql-cz?hl=cs
- http://pgxn.org
- https://stackoverflow.com/questions/tagged/postgresql

Instalace ve zkratce

Instalace z komunitního repozitáře

```
sudo dnf install postgresql
sudo dnf install postgresql15-server
sudo dnf install postgresql15-contrib
sudo dnf install pgadmin4
# apt install postgresql postgresql-client \
      postgresql-contrib

sudo dnf systemctl edit postgresql-15
sudo GSETUP_INITDB_OPTIONS="-k" \
     usr/pgsql-15/bin/postgresql-15-setup initdb

# pg_dropcluster --stop 15 main
# pg_createcluster --locale de_DE.UTF-8 --start 15 main

sudo systemctl enable postgresql-15
sudo systemctl start postgresql-15
```

Instalace ve zkratce

Registrace uživatele pavel

```
[pavel@localhost ~]$ su
[root@localhost pavel]# su postgres
[postgres@localhost pavel]# createuser --interactive pavel
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) y
Shall the new role be allowed to create more new roles? (y/n) y
CREATE ROLE


postgres=# SELECT name, setting FROM pg_settings
               WHERE name IN ('data_directory','config_file');
      name       |                 setting
-----------------+----------------------------------------
 config_file     | /usr/local/pgsql/data/postgresql.conf
 data_directory | /usr/local/pgsql/data
(2 rows)
```

Instalace ve zkratce

Postinstalační kontrola, nastavení locales

```
[pavel@localhost ~]$ psql postgres
psql-15 (15.4)
Type "help" for help.

postgres=> SHOW lc_collate;
 lc_collate
-------------
 cs_CZ.UTF-8
(1 row)

postgres=> SELECT upper('žlutý kůň');
   upper
-----------
 ŽLUTÝ KŮŇ
(1 row)
```

Poznámka

Pro Microsoft Windows je collate *Czech, Czech Republic*

Porovnání o.s. SQL RDBMS

Silná čtyřka

Kandidáti

- *Firebird* vznikl v reakci na nejasnou licenční politiku Borlandu v roce 2000 na základě otevřených kódů InterBase 6.0.
- *MySQL* vznikl jako náhrada systému mSQL v roce 1995 ve fy.TcX (Švédsko), 2008 Sun, 2009 Oracle.
- *PostgreSQL* vznikl doplněním jazyka SQL do RDBMS Postgres (Berkeley,95).
- *SQLite* je reakcí Richarda Hippa na velké RDBMS (USA, 2000).

Kritéria

- Použitelnost pro OLTP systémy
- Použitelnost pro web
- Použitelnost pro embedded systémy
- Restriktivnost licence

Porovnání o.s. SQL RDBMS

Použitelnost pro OLTP systémy

PostgreSQL	**MySQL (InnoDB)**
Firebird	**SQLite**

Požadavky

- spolehlivost (kritéria ACID)
- vysoká průchodnost v silně konkurenčním prostředí, déle trvající transakce,
- zajištění referenční, doménové a entitní integrity,
- maximální možná podpora ANSI SQL (plná podpora příkazu SELECT, pohledy, uložené procedury, triggery).

Porovnání o.s. SQL RDBMS

Použitelnost pro web

PostgreSQL	**MySQL (MyISAM)**
Firebird	**SQLite**

Požadavky

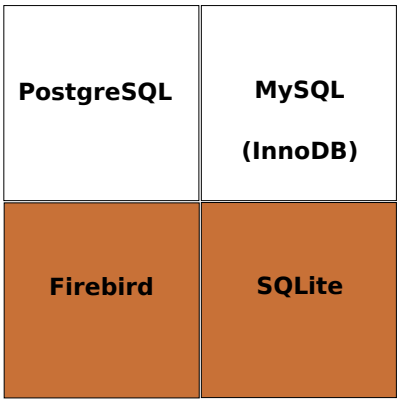
- vysoká průchodnost
- velmi rychlé zpracování jednodušších příkazů,
- rychlé vytvoření spojení mezi databází a klientem, extrémně krátké transakce,
- databáze musí zvládnout extrémní intenzitu příkazů,
- databáze musí se musí vyrovnat s intenzivními změnami obsahu tabulek.

Poznámka

OLTP aplikace, byť s tenkým klientem, zůstává stále OLTP aplikací.

Porovnání o.s. SQL RDBMS

Použitelnost pro Embedded systémy

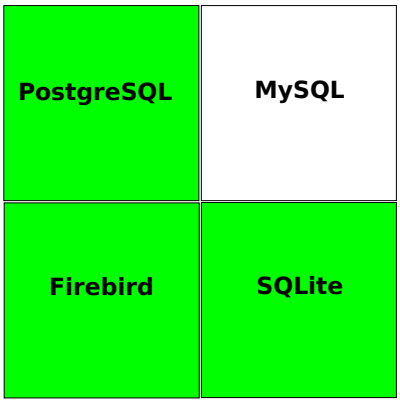


Požadavky

- minimální systémové nároky,
- minimalistická instalace a šíření (jeden datový soubor, jedna knihovna),
- spolehlivost. schopnost automatického zotavení se z chyb,
- minimální požadavky na provozní administraci,
- minimální závislosti.

Porovnání o.s. SQL RDBMS

Restriktivnost licence



Restrikce

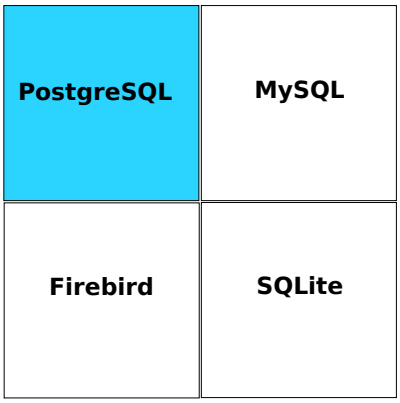
- používání v nekomerčním prostředí
- používání v komerčním prostředí,
- šíření pro GPL aplikace,
- šíření pro ne GPL aplikace,
- komerčního šíření modifikovaného kódu.

Licence

1. Public Domain (SQLite)
2. BSD licence (PostgreSQL)
3. licence IPL (Firebird)
4. Duální, GPL (pro o.s. projekty a vlastní i komerční použití) a komerční pro ostatní

Porovnání o.s. SQL RDBMS

Hodnocení PostgreSQL



Doporučení

Databázový systém PostgreSQL je navržen zejména pro OLTP prostředí (podnikové aplikace). Jako embedded databázi jej prakticky nasadit nelze. V prostředí webu je vhodné nasadit PostgreSQL ***v kombinaci*** (pro netransakční data - např. www sessions, logy, atd) s *memcached*, *MySQL* nebo *SQLite*.

Varování

Nevhodný výběr DBMS může vést ke zvýšeným nákladům na hw, administraci nebo k zbytečně komplikovanému a nespolehlivému řešení.

Porovnání o.s. RDBMS

Hodnocení PostgreSQL

Silné stránky

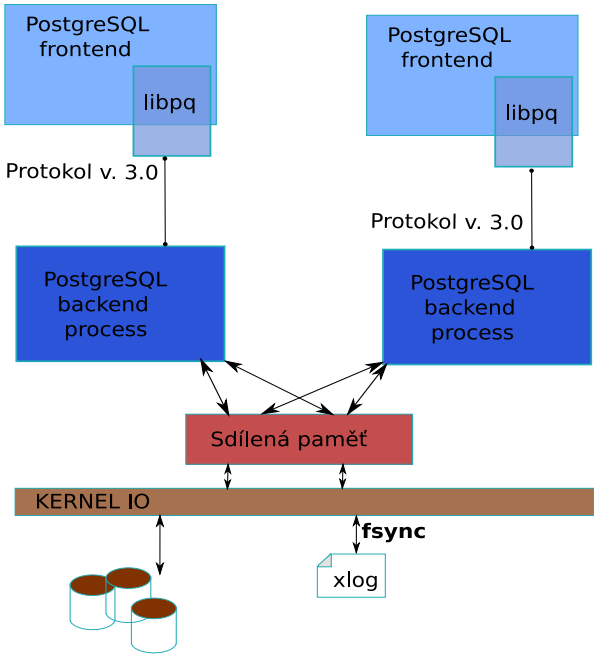
- podpora ANSI SQL 200x,
- sofistikovaný algoritmus získání optimálního prováděcího plánu,
- licence BSD,
- snadné doplňování funkcionality,
- bohatá nabídka vestavěných datových typů,
- řada kvalitních volně dostupných doplňků,
- silný interní PL jazyk s vynikající diagnostikou chyb,
- podpora externích PL jazyků,

Slabé stránky

- nutnost pravidelně spouštět příkaz VACUUM,
- díky MGA, v některých případech, limitující rychlost zápisu a čtení,
- nelze sdružovat do clusterů,
- replikační systém umožňuje pouze master-slave replikaci,

PostgreSQL v kostce

Architektura

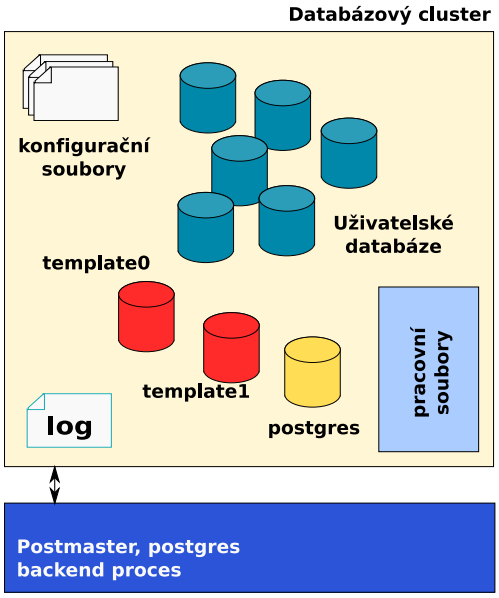


Popis

- klasický klient-server systém, komunikace prostřednictvím UDF, TCP/IP nebo SSL protokolu,
- víceprocesový systém, nové procesy vznikají pří vytvoření spojení,
- vždy jedno spojení, jeden proces,
- existuje možnost poolingu spojení.

PostgreSQL v kostce

Databázový cluster



Cluster obsahuje

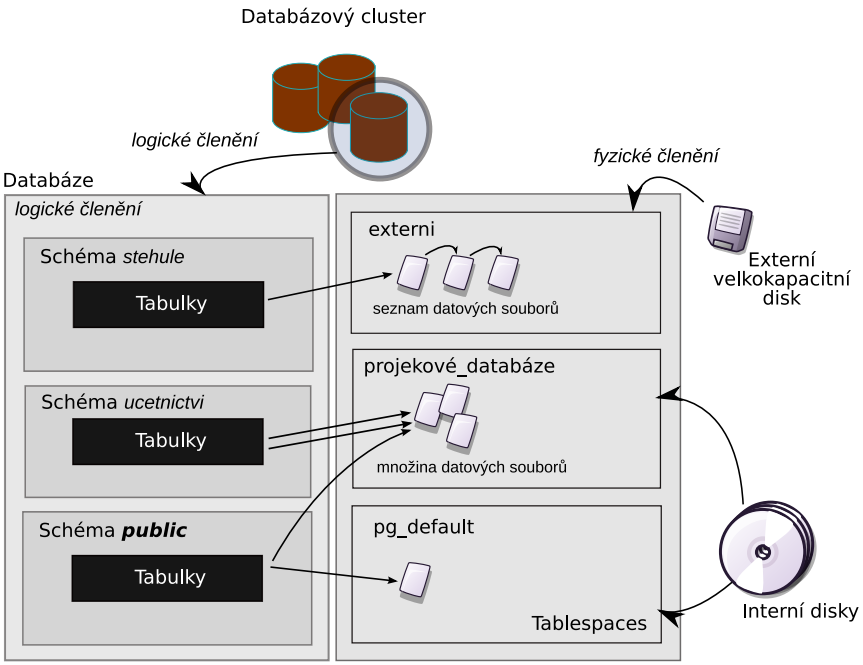
- datové soubory,
- pracovní soubory,
- konfigurační soubory,
- seznam uživatelů a zvolené LOCALES.

Mapování na sys. zdroje

- cluster → adresář,
- databáze → adresář,
- tabulka → soubor.

PostgreSQL v kostce

Logické členění/fyzické členění databáze



Inicializace clusteru

Správa datového adresáře

Inicializace

Slovo cluster má v PostgreSQL význam prostoru pro všechny databáze, ke kterým lze přistupovat určenou IP adresou a portem. Všechny databáze v clusteru sdílí konfiguaraci a uživatele. Na jednom počítači lze provozovat více clusterů stejných nebo různých verzí PostgreSQL.

```
initdb [OPTION]... [DATADIR]
    -E, --encoding=ENCODING  určí kódování
    --locale=LOCALE          určí locales
    -k, --data-checksums     kontrolní součty pro stránky
```

Poznámky

- v clusteru nic nemazat (`pg_resetxlog`)
- z clusteru nekopírovat datové soubory (nepřenosné)
- lze kopírovat celý adresář clusteru (zastavené PostgreSQL)
- lze kopírovat celý adresář clusteru (*aktivní export write ahead logu*)

Údržba databáze

Správa datového adresáře

Umístění datového adresáře

liší se dle zvyklostí konkrétních distribucí:

default `/usr/local/pgsql/data`, vytváří se ručně

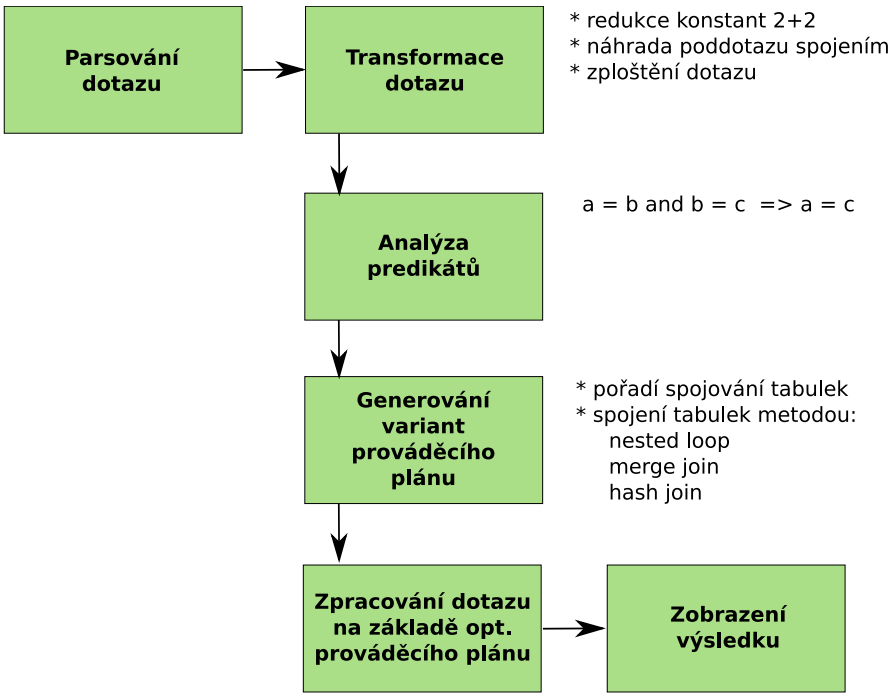
Red Hat `/var/lib/pgsql/data`, vytváří se automaticky, při prvním startu

Kontrola korektního chování *LOCALE*

Okamžitě po instalaci ověřte, zda je korektně podporováno národní prostředí. Nejlépe `SELECT upper('příliš žluťoučký kůň')`. Vybrané *locale* clusteru se musí shodovat s kódováním databáze, např. pro UTF8 musíme používat *locale* `cs_CZ.UTF8`.

PostgreSQL v kostce

Zpracování dotazu



Požadavky na transakční databázové systémy

tzv. ACID kritéria

Transakční systém

Za transakční systém považujeme každý systém, který splňuje kritéria ACID

Kritéria ACID

Atomičnost V rámci transakce se provedou vždy všechny příkazy nebo žádný.

Konzistence Transakce převádí data vždy z jednoho konzistentního stavu do druhého. Uvnitř transakce tato podmínka neplatí.

Izolace Transakce se navzájem neovlivňují.

Trvanlivost Pokud je transakce potvrzená, pak jsou změny trvalé.

Multigenerační architektura

Multi Value Concurency Control

Idea

Místo přepisu záznamů záznamy kopírujeme. Viditelné jsou pouze ty verze záznamů, které se váží na potvrzené transakce nebo nejnovější verze vytvořené aktuální transakcí.

Motivace

Důvodem implementace jsou ACID požadavky *konzistence* a *izolace*. Systém musí dokázat odstranit změny způsobené nedokončenou nebo přerušenou transakcí. Systém musí zajistit izolaci transakcí, tj. z transakce nikdy nejsou viditelné změny jiných nepotvrzených transakcí.

Multigenerační architektura

Multi Value Concurency Control

Výhody

- Minimalizace případů, kdy je nutné použít zámek. Vždy je k dispozici původní varianta záznamů, tudíž nepotvrzené transakce neblokují čtení. ***Vysoká prostupnost v konkurenčním prostředí.***
- Náročnost operací ROLLBACK a COMMIT nezávisí na objemu odvolávaných nebo potvrzovaných dat. Stačí pouze potvrzení nebo nepotvrzení transakce v seznamu provedených transakcí. Nikdy nedochází k přesunu dat.

Nevýhody

- ***Nutnost odstraňovat nedostupné záznamy.*** Databáze bobtná.
- Vzhledem k složitějšímu způsobu ukládání dat několikanásobně až řádově pomalejší čtení a zápis do databáze.
- Chybí podpora tzv. špinavého čtení.

Datové typy bez limitů - TOAST

The Oversized Attribute Storage Technique

Motivace

Odstranění limitu 8KB na max. velikost záznamu (velikost datové stránky). Úsporné ukládání textových dat.

Řešení

U položek delších 32 byte se zjišťuje efektivita komprimace. Pokud je účinnost větší než 25%, data se ukládají do tabulky TOAST komprimovaně. Podle typu uložiště (EXTERNAL, EXTENDED) se ukládaná data rozdělí do posloupnosti 2KB bloků a uloží do tabulky TOAST. Vše je pro uživatele ***transparentní***.

Efekty

- Maximální velikost textových typů je 1GB (praktické je 6-10MB).
- Úsporné ukládání textových dat (např. 50% u textů v HTML).
- Náročnější (byť transparentní) přístup k položkám delších 2KB.

PostgreSQL v kostce

Varianty uložení dat - plain, main, extended, external

Strategie TOAST

V případě, že řádek přesáhne určitou velikost (typicky 2KB), dochází k přesouvání nejdelších hodnot do přidružené tabulky TOAST, a to tak dlouho, dokud se nedosáhne požadované velikosti (2KB).

Varianty

Plain - blokuje komprimaci, blokuje ukládání mimo tabulku (max. velikost 8KB), výchozí pro typy s fixní délkou,

Main - data jsou komprimována, a pokud je to možné, tak uložena lokálně,

Extended - data jsou komprimována, a ukládána mimo tabulku (out-of-line) (pokud je to nutné), výchozí varianta pro tzv. varlena typy,

External - data jsou ukládána mimo tabulku - nekomprimována (rychlejší operace substring - spec. optimalizace),

Spolehlivost a výkon - WAL

Write ahead logging

Motivace

Je řešením ACID požadavku na trvanlivost. Každá změna datových souborů musí být jištěna předchozím zápisem do transakčního logu.

Efekty

- V případě výpadku lze databázi obnovit z transakčního logu, tj. je zajištěna konzistence datových souborů a indexů.
- Při potvrzení transakce je nutné provést operaci *fsync* pouze na transakčním logu (nikoliv nad datovými soubory). Výsledkem je zásadní zvýšení výkonu. Sdílením transakčního logu dochází k další redukci fsynců.
- Transakční log lze exportovat na bezpečné médium. tj. on-line zálohování a PITR (Point In Time Recovery).

Nutné zlo, příkaz VACUUM

příkaz ANALYZE

Aktualizuje datové statistiky použité optimalizátorem dotazů (velikost tabulek, histogramy hodnot pro jednotlivé sloupce).

příkaz VACUUM

Odstraňuje nedostupné (mrtvé) verze záznamů z datových souborů. Uvolňuje prostor jimi obsazený a zároveň zrychluje přístup k dostupným záznamům (při čtení se nepřeskakují mrtvé záznamy).

Varování

V případě, že podíl mrtvých variant záznamů dosáhne 30%, dochází k znatelnému zpomalení všech operací nad tabulkou. Pokud nedojde k provedení příkazu VACUUM může být databáze po určitém čase prakticky nefunkční, přičemž generuje značnou zátěž serveru. Obranou je pravidelné spouštění příkazu VACUUM nebo aktivace procesu pg_autovacuum.

Nutné zlo, příkaz VACUUM

Varianty příkazu VACUUM

Variace příkazu

- *VACUUM*, které nevyžaduje žádný zámek tabulky.
- *VACUUM ANALYZE* zároveň provede aktualizaci statistik.
- *VACUUM FULL* provede reorganizaci záznamů na disku (zaplněním mezer). Vyžaduje exkluzivní zámek nad tabulkou.
- *VACUUM FREEZE* provede ”zmražení” všech aktuálních ***živých*** záznamů a resetuje čítač transakcí.

Zmražení záznamu

Každý záznam obsahuje 4byte identifikátor transakce, která jej vytvořila. Při vyčerpání (přetečení) 4byte prostoru hrozí kolaps MGA mechanismu. ”Zmražení” záznamu znamená, že jeho *transaction ID* je nastaveno na předdefinovanou hodnotu *FroozenXID*.

Nutné zlo, příkaz VACUUM

Řešení

Periodické spouštění příkazu VACUUM

- Pro:
  - přesné načasování spuštění příkazu.
- Proti:
  - obtížně se odhaduje optimální frekvence spouštění příkazu,
  - za aktivaci zodpovídá externí služba, což může způsobovat provozní a administrativní problémy.

Proces *pg_autovacuum*

- Pro:
  - relativně přesné načasování spouštění příkazu.
- Proti:
  - pokud se spustí ”nevhodně”, způsobí snížení výkonu systému (režie),
  - vyžaduje zapnutí provozních statistik (20% režie).

Údržba databáze

Opakující se činnosti

Pravidelné

- ***1x denně*** `VACUUM ANALYZE` (`cron`, `autovacuum`),
- 1x měsíčně `REINDEX` databáze nebo alespoň nejčastěji modifikovaných tabulek,

Nepravidelné

- pokud dochází vyhrazený prostor na disku pro data `VACUUM FULL`,
- pokud hrozí přetečení čítače transakcí (varování v logu) `VACUUM FREEZE` (1x za několik let),
- analýza pomalých dotazů (limit 200ms) a jejich eliminace přidáním nových indexů.
- čištění datového schéma (nutno zálohovat a dokumentovat)
  - odstranění nepoužívaných tabulek (`pg_stat_all_tables`),
  - odstranění nepoužívaných indexů (`pg_stat_all_indexes`).

Každý index zabírá prostor na disku a zpomaluje operace `INSERT`***,*** `UPDATE`***,*** `DELETE`. Proto indexy vytváříme pouze tehdy, když mají nějaký efekt.

Údržba databáze

Zastavení, spuštění PostgreSQL

Start, Reload, Restart serveru

- `/etc/init.d/postgres (start|reload|restart|stop|status)`
- `pg_ctl` lze specifikovat režimy ukončení
  - `smart` čeká, až se odhlásí všichni klienti,
  - `fast` nečeká na odhlášení klientu, provede úplný proces ukončení,
  - `immediate` skončí okamžitě s důsledkem obnovy po nekorektním ukončení při příštím startu.

Preferujeme co nejšetrnější možnou metodu. V případě, podivného chování jednoho klientského procesu, lze zaslat signál `sigint` obslužnému procesu klienta.

Ukončení klienta z prostředí SQL

1. získání *pid* problémového klienta

```
select procpid, usename, current_query from pg_stat_activity;
procpid | usename |  current_query
  10144 | root    |  select fce();
```

2. odstranění procesu `select pg_cancel_backend(10144);`

Údržba databáze

Zastavení, spuštění PostgreSQL

Obnova po havárii

Pokud server nebyl ukončen korektně, je možné, že v paměti zůstanou servisní procesy PostgreSQL. Ty je nutné před opětovným spuštěním serveru ukončit. Výjmečně je nutné vyčistit sdílenou paměť příkazem `ipcclean`. Také je nutné explicitně odstranit soubor `dbcluster/postmaster.pid`. Za normálních okolností se spustí automaticky proces obnovy databáze na základě údajů uložených ve *write ahead logu*.

Doporučení

Je celkem na místě ověřit integritu databáze dumpem databáze. V případě problémů

- reindexace databáze včetně systémových tabulek
- obnova ze zálohy
- identifikace poškozených řádků a jejich odstranění. Příznakem poškozené databáze je pád serveru při sekvenčním čtení.

Rozšiřitelnost PostgreSQL

Vlastní funkce

Návrh vlastních funkcí

- C, PL/pgSQL, PL/SQL, PL/Perl, PL/Python, PL/Java, PL/Php, PL/R
- Podpora OUT parametrů
- Podpora Set Returning Functions (výsledkem je tabulka)
- Podpora ošetření chyb
- Podpora polymorfismu

```
CREATE OR REPLACE FUNCTION
plvstr.swap(str text, replace text, start int, length int)
RETURNS text
AS '$libdir/orafunc','plvstr_swap'
LANGUAGE c STABLE;
```

Rozšiřitelnost PostgreSQL

Vlastní funkce, ukázka SQL funkce

Funkce sign

Funkce *mysign* vrací hodnotu -1 pro záporný argument, hodnotu 0 pro nulu a hodnotu 1 pro kladné nenulové číslo.

```
CREATE OR REPLACE FUNCTION mysign(a numeric)
RETURNS numeric AS $$
SELECT CASE WHEN $1 < 0 THEN -1::numeric
            WHEN $1 > 0 THEN  1::numeric
            ELSE 0::numeric END;
$$ LANGUAGE sql;
```

Rozšiřitelnost PostgreSQL

Vlastní funkce, ukázka funkce v Perlu

Funkce rsplit

Funkce *rsplit* vrací vrací pole řetězců identifikovaných regulárním výrazem. Např. výsledkem `rsplit('123456 22','([0-9]+)'` je pole {123456,22}.

```
CREATE OR REPLACE FUNCTION rsplit(text, text)
RETURNS text[]
AS $$
     my @result = ($$_[0] =~ /$_[1]/g);
     return \@result;
$$ LANGUAGE plperl IMMUTABLE STRICT;

-- totez v 8.3.
SELECT ARRAY(SELECT re[1]
                FROM regexp_matches('123456 789',e'\\d+','g') re);
```

Rozšiřitelnost PostgreSQL

Vlastní operátor, ukázka operátoru div pro typ interval

Operace dělení pro typ interval

Výsledkem '1 hours' / '10 minutes je číselná hodnota 6.

```
CREATE FUNCTION div(interval, interval)
RETURNS double precision AS $$
  SELECT EXTRACT(epoch FROM $1) / EXTRACT(epoch from $2);
$$ LANGUAGE sql;

CREATE OPERATOR / (
  PROCEDURE = div,
  LEFTARG = interval, rightarg = interval);
```

Rozšiřitelnost PostgreSQL

Vlastní operátor, ukázka operátoru div pro modifikovaný operátor nerovnosti

Operátor nerovno inertní hodnotě NULL

Chceme, aby platilo i pro NULL, že NULL <> konstanta

```
CREATE OR REPLACE FUNCTION cmp_ne_spec(anyelement, anyelement)
RETURNS boolean AS $$
  SELECT $1 IS NULL OR $2 IS NULL OR $1 <> $2;
$$ LANGUAGE sql;

CREATE OPERATOR <<>> (
  PROCEDURE=cmp_ne_spec,
  LEFTARG=anyelement,
  RIGHTARG=anyelemeny);
```

Rozšiřitelnost PostgreSQL

Vlastní operátor, ukázka vlastní automatické konverze z int na timestamp

Automatická konverze integer na timestamp

Převádí unix. čas (počet sec od 1.1.1970) na PostgreSQL timestamp. Např. výsledkem 0::timestamp je 1.1.1970 00:00:00.

```
CREATE FUNCTION to_timestamp(integer)
RETURNS timestamp with time zone AS $$
  SELECT to_timestamp($1::float8);
$$ LANGUAGE sql;

CREATE CAST (integer AS timestamp with time zone)
  WITH FUNCTION to_timestamp(integer)
  AS IMPLICIT;
```

Základní příkazy pro správu PostgreSQL

Seznam příkazů

Pro přístup a používání databáze

createdb	založí novou databázi,
dropdb	odstraní existující databázi,
psql	sql konzole umožňující zadávání SQL příkazů.

Pro administraci

vacuumdb	spouští vacuum databáze,
reindexdb	znovu vytvoří indexy v databázi,
createuser	přidá uživatele do databázového clusteru,
dropuser	odstraní uživatele z databázového clusteru,
oid2name	zobrazuje číselný identifikátor jako text,
pg_dump	vytvoří dump databáze,
pg_dumpall	vytvoří dump celého databázového clusteru,
initdb	inicializuje databázový cluster,
pgbench	TPC-B test výkonu (hrubé ověření instalace),
pg_restore	obnoví databázi ze zálohy,
pg_basebackup	vytvoří fyzický klon (online zálohu) clusteru.

Základní příkazy pro správu PostgreSQL

Seznam metapříkazů psql

Metapříkazy

\h	nápověda k SQL příkazům,
\?	nápověda k metapříkazům,
\q	ukončení konzole,
\r	vyčištění vyrovnávací paměti textu dotazu,
\i	provedení SQL příkazů ze souboru,
\l	seznam databází,
\d	popis objektu,
\dt	seznam tabulek,
\df	seznam funkcí,
\dn	seznam schémat,
\x	přepínání mezi sloupcovým a řádkovým zobrazením,
\timing	přepíná zobrazení doby provádění dotazu,
\e	aktivuje externí editor,
tab	autocomplete,
ctrl+r	vyhledávání v historii.

Základní příkazy pro správu PostgreSQL

Ukázka psql konzole

```
[pavel@localhost ~]$ psql postgres
psql (9.0.3)
Type "help" for help.

postgres=> \h create trigger
Command:     CREATE TRIGGER
Description: define a new trigger
Syntax:
CREATE TRIGGER name { BEFORE | AFTER } { event [ OR ... ] }
    ON table [ FOR [ EACH ] { ROW | STATEMENT } ]
    EXECUTE PROCEDURE funcname ( arguments )

postgres=> \d foo
           Table ``public.foo''
 Column |       Type        | Modifiers
--------+-------------------+-----------
 i      | integer           |
 a      | character varying |

postgres=>
```

Základní příkazy pro správu PostgreSQL

Ukázka psql konzole

```
postgres=> select current_time;
       timetz
--------------------
 13:45:08.833422+02
(1 row)

postgres=> CREATE OR REPLACE FUNCTION hello(varchar)
postgres-> RETURNS varchar
postgres-> AS $$
postgres$$> BEGIN
postgres$>   RETURN 'Hello ' || $1;
postgres$> END;
postgres$> $$ LANGUAGE plpgsql stable;
CREATE FUNCTION
postgres=>
postgres=> select hello('world');
    hello
-------------
 Hello world
(1 row)

postgres=> \q
[pavel@localhost ~]$$
```

Základní příkazy pro správu PostgreSQL

Systémový katalog - použití metapříkazů

```
postgres=# \dt
           List of relations
 Schema |     Name     | Type  | Owner
--------+--------------+-------+-------
 public | accounts     | table | pavel
 public | branches     | table | pavel
 public | comp         | table | pavel
 public | dual         | table | pavel

postgres=# \d foo
      Table "public.foo"
 Column |  Type   | Modifiers
--------+---------+-----------
 a      | integer |
```

Základní příkazy pro správu PostgreSQL

Systémový katalog - dohledání zdrojů

```
[pavel@okbob-bb ~]$ psql -E postgres

postgres=# \dt
******** QUERY **********
SELECT n.nspname as ``Schema'',
  c.relname as ``Name'',
  CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view'
       WHEN 'i' THEN 'index' WHEN 'S' THEN 'sequence'
       WHEN 's' THEN 'special' END as ``Type'',
  r.rolname as ``Owner''
FROM pg_catalog.pg_class c
     JOIN pg_catalog.pg_roles r ON r.oid = c.relowner
     LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r','')
      AND n.nspname NOT IN ('pg_catalog', 'pg_toast')
  AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
**************************

           List of relations
 Schema |     Name     | Type  | Owner
--------+--------------+-------+-------
 public | accounts     | table | pavel
 public | branches     | table | pavel
```

Základní příkazy pro správu PostgreSQL

Systémový katalog - použití informačních schémat

```
postgres=# select table_name, table_schema
              from information_schema.tables
             where table_schema = 'public';
  table_name  | table_schema
--------------+--------------
 test         | public
 pg_ts_dict   | public
 pg_ts_parser | public
 pg_ts_cfg    | public
 pg_ts_cfgmap | public
 branches     | public
 tellers      | public
 history      | public
 accounts     | public
```

Export, import dat

Možnosti

SQL dump tabulek a struktury

Systémovým příkazem pg_dump dokážeme exportovat tabulku (nebo více tabulek) a to jak data, tak strukturu. Výsledný soubor obsahuje SQL příkazy. Data lze uložit jako sadu příkazů INSERT nebo jako jeden příkaz COPY.

COPY na serveru

Tento příkaz vytvoří (přečte) textový soubor (hodnoty oddělené čárkou nebo tabelátorem) uložený na serveru. Při zpracování nedochází k přenosu dat po síti. Musíme mít k dispozici prostor na serveru přístupný pro uživatele postgres.

COPY z klienta

Obdoba příkazu COPY implementovaná jako příkaz konzole psql. Čte (ukládá) soubory na stranně klienta, zajišťuje přenos dat po síti, a zpracování na straně serveru. Formát souboru je stejný jako na u příkazu COPY na serveru.

Export, import dat

Možnosti

file_fdw

Použití tzv externího datového zdroje - *foreign Data Wrapper* - umožňuje dotazy na data uložená mimo SQL server. Jedním z dostupných ovladačů je *file_fdw* umožňující číst data ze souboru (ve stejném formátu jako příkaz COPY). K dispozici jsou ovladače pro MySQL, Oracle, ODBC, .. Data lze pouze číst.

Export, import dat

pg_dump

```
pg_dump [OPTION]... [DBNAME]
      -a, --data-only           pouze data
      -c, --clean               předřadí příkazy pro zrušení
                                objektů
      -C, --create              vloží příkazy k vytvoření
                                objektů
      -d, --inserts             použije INSERT místo COPY
      -E, --encoding=ENCODING   použije určené kódování
      -s, --schema-only         pouze schéma
      --disable-triggers        po dobu načítání dat blokuje
                                triggery
      -t, --table=TABLE         pouze určitou tabulku
```

Export, import dat

COPY

```
COPY tablename [ ( column [, ...] ) ]
    (FROM|TO) { 'filename' | (STDIN|STDOUT) }
    [ [ WITH ]
          [ BINARY ]
          [ HEADER ]
          [ OIDS ]
          [ DELIMITER [ AS ] 'delimiter' ]
          [ NULL [ AS ] 'null string' ]
          [ CSV [ HEADER ]
                [ QUOTE [ AS ] 'quote' ]
                [ ESCAPE [ AS ] 'escape' ]
                [ (FORCE NOT NULL column [, ...]|
                   FORCE QUOTE column [, ...]) ]
```

Export, import dat

file_fdw

```
postgres=# CREATE EXTENSION file_fdw;
CREATE EXTENSION

postgres=# CREATE SERVER file_server
              FOREIGN DATA WRAPPER file_fdw;
CREATE SERVER

postgres=# CREATE FOREIGN TABLE tbl (a int, b int, c int)
              SERVER file_server
             OPTIONS (format 'csv', filename '/tmp/hodnoty.csv');
CREATE FOREIGN TABLE

postgres=# SELECT * FROM tbl where a > 10;
 a  | b  | c
----+----+----
 40 | 50 | 60
 70 | 80 | 90
(2 rows)
```

Export, import dat

Efektivita

Přehled jednotlivých metod

Uvedená tabulka obsahuje údaje o importu jednoho miliónu řádek jednosloupcové celočíselné tabulky (testováno na notebooku Prestigio Nobile 156, P1.6, 500M).

Metoda	Velikost	Čas
INSERTS (autocommit on)	37,8 M	10 min
INSERTS (autocommit off)	37,8 M	2.2 min
COPY	6,8 M	10 sec
\COPY (UDP)	6.8 M	10 sec
COPY BINARY	10 M	7 sec
COPY (+ 1 index)	6.8 M	17 sec

Závěr

- Preferovat COPY,
- Zrušit všechny indexy (nejsnáze pomocí SQL procedury),
- zablokovat triggery (`ALTER TABLE name DISABLE TRIGGER ALL`).

Zálohování, obnova databáze

Přehled

Přehled technik zálohování

K dispozici jsou tři způsoby zálohování:

- *SQL dump* příkazy pg_dump, pg_restore. Data lze uložit jako SQL skript nebo v speciálním komprimovaném formátu.
- *záloha na úrovni souborového systému* ***Server musí být zastaven!*** Lze zálohovat a obnovovat pouze kompletní db. cluster.
  `tar -cf backup.tar /usr/local/pgsql/data`
- *online zálohování* je založeno na tzv. *write ahead logu* (WAL), což je soubor do kterého se zapisují všechny změny v datech ještě předtím než se zapíší do datových souborů. Primárně tento log slouží k obnově databáze po havárii, můžeme jej však exportovat, uložit a použít pro vytvoření zálohy.
  - jediné možné řešení průběžného zálohování,
  - náročné na diskový prostor,
  - náročné na administraci,
  - zálohování i obnova je velice rychlé,
  - záloha není kompatibilní mezi 32 a 64 bit platformami.

Zálohování, obnova databáze

Vytvoření zálohy exportováním WAL

Postup

1. V *postgresql.conf* nastavíme `archive_command`. Tento příkaz zajistí přenesení segment logu na bezpečné médium. PostgreSQL neodstraní segment, dokud příkaz neproběhne bezchybně.
   ```
   archive_command = 'cp -i "%p" /mnt/server/archivedir/"%f"'
   ```
2. Export logu aktivujeme voláním `select pg_start_backup('navesti')`. Jako návěstí můžeme použít libovolný řetězec, např. cesta k záloze.
3. V tomto případě můžeme bezpečně za chodu zkopírovat obsah dovohého adresáře PostgreSQL. Není třeba zálohovat adresář *pg_xlog*.
4. po dokončení kopírování deaktivujeme export logu voláním `SELECT pg_stop_backup()`. Export logu může běžet libovolnou dobu, což je základ průběžného zálohování.

Průběžné zálohování

Segment se exportuje po naplnění (16MB) nebo po přednastaveném časovém intervalu (8.2). Efektivně jej lze komprimovat.

Zálohování, obnova databáze

Obnova ze zálohy exportovaného WAL

Postup

1. Zazálohujte si aktuální cluster (včetně *pg_xlog*).
2. Překopírujte data ze zálohy (zazálohovaný datový adresář).
3. Upravte soubor *recovery.conf* a uložte jej v adresáři clusteru. Vzor naleznete v podadresáři *shared*. Minimální změnou je nastavení položky `archive_command`.
4. do nadresáře *pg_xlog* zkopírujte všechny nezazálohované soubory z adresáře *pg_xlog* (to jsou WAL segmenty, které vznikly po deaktivaci exportu).
5. Nastartujte server. Při startu se automaticky spustí proces obnovy na základě WAL.

Správa uživatelů

createuser

```
Usage:
  createuser [OPTION]... [ROLENAME]

Options:
  -s, --superuser           role will be superuser
  -S, --no-superuser        role will not be superuser
  -d, --createdb            role can create new databases
  -D, --no-createdb         role cannot create databases
  -r, --createrole          role can create new roles
  -R, --no-createrole       role cannot create roles
  -l, --login               role can login (default)
  -L, --no-login            role cannot login
  -i, --inherit             role inherits privileges of roles it is a
                            member of (default)
  -I, --no-inherit          role does not inherit privileges
  -c, --connection-limit=N  connection limit for role (default: no limit)
  -P, --pwprompt            assign a password to new role
  -E, --encrypted           encrypt stored password
  -N, --unencrypted         do not encrypt stored password
  -e, --echo                show the commands being sent to the server
  -q, --quiet               don't write any messages
```

Správa uživatelů

CREATE ROLE

```
Command:     CREATE ROLE
Description: define a new database role
Syntax:
CREATE ROLE name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | CREATEUSER | NOCREATEUSER
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | IN ROLE rolename [, ...]
    | ROLE rolename [, ...]
    | ADMIN rolename [, ...]
    | USER rolename [, ...]
    | SYSID uid
```

Správa uživatelů

Výklad

Pravidla chování rolí I.

- Závislosti mezi rolemi tvoří orientovaný graf, který nesmí obsahovat cyklus (Pavel může získat práva Tomáše, zároveň ale Tomáš nemůže získat práva Pavla).
- Uživatel získá práva rolí, jejichž je členem (ke kterým má přístup, které může převzít).

  ```
  CREATE ROLE tom_a_pavel IN ROLE tom, pavel;
  ```

  Role *tom_a_pavel* může převzít roli Tomáše nebo Pavla (je to nadřazená role těmto rolím), a má práva jako Tomáš a Pavel dohromady.
- Roli můžeme definovat také tak, že určíme kdo tuto roli může používat.

  ```
  CREATE ROLE developer ROLE tom, pavel;
  ```

  Roli *developer* může použít jako Tomáš tak Pavel. Pokud má role atribut `INHERIT` tak automaticky získává práva všech rolí, jejichž je členem (které může použít). Bez tohoto atributu vývojář musí explicitně aktivovat roli příkazem `SET ROLE developer`.

Správa uživatelů

Výklad

Pravidla chování rolí II.

- Uživatel může změnit vlastnictví objektů ke kterým má práva příkazem:

  ```
  ALTER TABLE objekt OWNER TO developer;
  ```

  ale nemůže se vzdát svých práv, tj. nemůže změnit vlastnictví tak, aby přišel o objekt (nelze databázový objekt darovat někomu neznámému s nímž nemám nic společného.

Rekapitulace

`CREATE ROLE` vytvoří novou roli.
`GRANT co TO komu` delegování určitého práva roli.
`GRANT r1 TO r2` role r2 získává stejná práva jako má role r1.
`REVOKE` odejmutí práv.
`ALTER TABLE .. OWNER TO ..` změna vlastnictví objektu.
\dg zobrazení rolí a členství v psql.

Konfigurace databáze

Volba souborového systému

Vliv souborového systému na výkon databáze

Nelze obecně říci, který souborový systém je optimální. Při **umělých** testech bylo pořadí souborových systémů následující: *JFS*, *ext2*, *Reiser3*, *ext3*, *XFS*. Rozdíl mezi nejpomalejší a nejrychlejší testovací konfigurací byl 30% (což se nebere jako natolik významná hodnota, aby se o tom příliš diskutovalo). U ”high” řadiču je nutné explicitně povolit write cache (baterií zálohované řadiče).

Závěr

- používejte takový souborový systém, který je pro vás důvěryhodný (RAID 10),
- na UNIXech používejte mount parametr `noatime`,
- pokud jste jištěni UPS, lze pro *ext3* použít mount parametr `data=writeback`. výkonově se dostanete na úroveň *ext2*, v žádném případě se nedoporučuje změnit konfigurační parametr *fsync* na `off`,
- pokuste se umístit WAL na jiný nejlépe RAID 1 disk než data (symlink) .
- verifikujte konfiguraci testem *pgbench*, který by měl zobrazovat rámcově srovnatelné hodnoty s jinou instalací PostgreSQL na podobném hw.

Konfigurace databáze

Přidělení paměti

Strategie

PostgreSQL má mít přiděleno maximum paměti, aniž by zpomalila o.s. Přidělení paměti je určeno hodnotami určitých parametrů v *postgresql.conf*. Pouze parametr `work_mem` lze nastavovat dynamicky. Neexistuje úplná shoda ohledně doporučených hodnot.

postgresql.conf I.

`shared_buffers` velikost cache pro systémové objekty [32..2048] MB (6..25%).

`work_mem` velikost alokované pracovní paměti ***pro každé spojení***, omezuje použití diskové mezipaměti při operaci sort, určuje maximální velikost hash tabulek při operaci hash join, lze ji nastavit dynamicky před náročným dotazem [1..10] MB.

Konfigurace databáze

Přidělení paměti

postgresql.conf I.

`maintenance_work_mem` velikost paměti používané pro příkazy jako jsou `VACUUM` nebo `CREATE INDEX`. Jelikož není pravděpodobné, že by došlo k souběhu provádění těchto příkazů lze bezpečně tuto hodnotu nastavit výrazně vyšší než je `work_mem`. Doporučuje se 32 ... 256MB nebo 50..75% velikosti největší tabulky.

Konfigurace databáze

Parametrizace plánovače dotazů

Poznámka

Až na výjimky, parametry týkající se výběru nejvhodnějšího způsobu provádění dotazu jsou určené pouze pro vývojáře PostgreSQL, a mají umožnit vzdálenou diagnostiku nahlášených problémů.

Konfigurace databáze

Parametrizace procesu *autovacuum*

Strategie

Častější provedení příkazu `VACUUM` než je nutné, je menším zlem než nedostatečně časté provádění tohoto příkazu. Spouští se periodicky (`cron`) nebo při překročení dynamické prahové hodnoty (*autovacuum*). Tato hodnota roste s velikostí tabulky.

postgresql.conf III.

`stats_row_level` aktualizace provozních statistik

`autovacuum` povolení samotného procesu (vyžaduje provozní statistiky)

Příkaz `VACUUM` se spustí, pokud počet modifikovaných řádků je větší než *autovacuum_vacuum_threshold* + (*autovacuum_vacuum_scale_factor* $*$ *velikost_tabulky*). Přibližně 20% počtu řádek v tabulce.
Příkaz `ANALYZE` se spustí, pokud počet modifikovaných řádků je větší než *autovacuum_analyze_threshold* + (*autovacuum_analyze_scale_factor* $*$ *velikost_tabulky*). Přibližně 10% počtu řádek v tabulce.

Monitorování databáze

Sledování aktivity

Pohled do systémových tabulek

- pohled *pg_stat_activity* obsahuje přehled činnosti přihlášených klientů.
- pohled *pg_stat_database* obsahuje počet přihlášených klientů k jednotlivým databázím.
- pohled *pg_stat_all_tables* obsahuje provozní statistiky tabulek.
- pohled *pg_stat_all_indexes* obsahuje provozní statistiky indexů.
- pohled *pg_locks* obsahuje seznam aktivních zámků.

postgresql.conf IV.

Sledování déletrvajících a chybných dotazů:

`log_min_error_statement = error` loguje všechny chybné SQL příkazy.

`log_min_duration_statement = 200` loguje všechny SQL příkazy prováděné déle než 200ms.

Na vytěžení údajů z logu lze použít tzv. PostgreSQL log analyzer *pgFouine*.

Monitorování databáze

Sledování aktivity

postgresql.conf V.

Sledování příkazů čekajících na uvolnění zámku:

`log_lock_waits = on` loguje všechny SQL příkazy čekající déle než je test na deadlock

Monitorování databáze

Monitorování podílu mrtvých záznamů a fragmentace indexu

```
  -- funkce pgstattuple a pgstatindex z balícku pgstattuple
postgres=# \x
Expanded display is on.
postgres=# select * from pgstattuple('foo');
-[ RECORD 1 ]------+------
table_len          | 8192
tuple_count        | 0
tuple_len          | 0
tuple_percent      | 0
dead_tuple_count   | 2
dead_tuple_len     | 93
dead_tuple_percent | 1.14
free_space         | 8064
free_percent       | 98.44

postgres=# select * from pgstatindex('foox');
-[ RECORD 1 ]------+-----
version            | 2
tree_level         | 0
index_size         | 8192
root_block_no      | 1
internal_pages     | 0
leaf_pages         | 1
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 0.93
leaf_fragmentation | 0
```

Monitorování databáze

Monitorování podílu mrtvých záznamů a fragmentace indexu

```
-- 30% mrtvých záznamů má znatelný vliv na rychlost provádění dotazů
SELECT schema, name, (st).tuple_count, pg_size_pretty((st).table_len) AS table_len,
       (st).dead_tuple_count, (st).dead_tuple_percent,
       pg_size_pretty((st).free_space) AS free_space
  FROM (
        SELECT n.nspname AS schema, c.relname AS name, pgstattuple(c.oid) AS st
           FROM pg_catalog.pg_class c
                LEFT JOIN
                pg_catalog.pg_namespace n
                ON n.oid = c.relnamespace
          WHERE c.relkind in ('r','') and pg_catalog.pg_table_is_visible(c.oid)
       ) s;

-- optimální využití je 60-70% koncových stránek
SELECT schema, "table", name, pg_size_pretty((st).index_size) AS index_size,
       (st).internal_pages AS internal, (st).leaf_pages as leaf,
       (st).empty_pages AS empty, (st).deleted_pages AS deleted ,
       (st).avg_leaf_density AS avg_leaf, (st).leaf_fragmentation AS leaf_frag,
       (st).tree_level
   FROM (
         SELECT schemaname AS schema, tablename AS table , indexname AS name,
                pgstatindex(schemaname || '.' || indexname) AS st
            FROM pg_indexes
        ) s;
```

Monitorování databáze

Monitorování obsahu sdílené vyrovnávací paměti

```
SELECT c.relname, count(*) AS buffers
   FROM pg_class c INNER JOIN pg_buffercache b
        ON b.relfilenode = c.relfilenode INNER JOIN pg_database d
        ON (b.reldatabase = d.oid AND d.datname = current_database())
  GROUP BY c.relname
  ORDER BY 2 DESC LIMIT 10;
```

```
              relname              | buffers
-----------------------------------+---------
 tenk2                             |     345
 tenk1                             |     141
 pg_proc                           |      46
 ...
(10 rows)
```

Poznámka

Obsah cache by měl být pokud možno stabilní. Pokud se často mění, znamená to intenzivní zatížení I/O operacemi (sekvenční čtení velkých tabulek). Může souviset s nevhodným nastavením `effective_cache_size`.

Instalace doplňků

Postup

Poznámka

PostgreSQL je navrhován minimalisticky, tj. co nemusí být částí jádra, přesouvá se do rozšiřujících (contrib) modulů. Příkladem může být *tsearch2*, *fuzzystrmatch*, *pgcrypto* nebo *pgbench*. Ukázka obsahuje instalaci doplňku *orafce*, což je sada funkcí inspirovaná knihovnou RDBMS Oracle.

1. Pokud jste v adresáři contrib `make; make install`,
2. V privátním adresáři `make USE_PGXS=1; make USE_PGXS=1 install`
3. Jako superuser provést příkaz `CREATE EXTENSION`
   ```
   CREATE EXTENSION orafce;
   ```
4. U některých doplňků je nutné explicitně zpřístupnit nové funkce příkazem `GRANT`. Tímto způsobem určujeme, kdo smí nové funkce používat.

pgbench

`pgbench` je jednoduchá klientská aplikace, která generuje unifikovanou zátež serveru PostgreSQL. V podstatě nelze jednoznačně interpretovat výslednou hodnotu udávající počet realizovaných zákaznických transakcí za vteřinu.

Instalace doplňků

Testování

Regresní test

Každý doplněk obsahuje sadu testu umožňujících alespoň rámcově ověřit funkčnost na dané platformě. Pokud selže regresní test, reportujte popis chyby a platformy správci testovaného doplňku.

```
make USE_PGXS installcheck
```

Instalace doplňků

Postup

```
postgres=# \dx
                 List of installed extensions
  Name   | Version |   Schema   |         Description
---------+---------+------------+------------------------------
 plpgsql | 1.0     | pg_catalog | PL/pgSQL procedural language
(1 row)

postgres=# select * from pg_available_extensions();
   name   | default_version |                  comment
----------+-----------------+-------------------------------------------
 xx       | 1.0             | data type for multidimensional cubes
 plperl   | 1.0             | PL/Perl procedural language
 unaccent | 1.0             | text search dictionary that removes accents
 plperlu  | 1.0             | PL/PerlU untrusted procedural language
 plpgsql  | 1.0             | PL/pgSQL procedural language
(5 rows)

postgres=# CREATE EXTENSION unaccent;
CREATE EXTENSION
```

Postup při přechodu na novou verzi

Plán

Pozn.

Při minoritní změně (změna za destinou tečkou) jsou verze datově kompatibilní (tudíž stačí pouze změnit binární soubory). Při aktualizaci mezi nekompatibilními verzemi je třeba provést dump databáze. V případě, kdy budeme migrovat na novější verzi je doporučováno, aby dump byl provedem příkazem `pg_dump` z novější verze (tj. nejdříve aktualizujeme klientské aplikace, poté server ... všechny PostgreSQL aplikace se dokážou připojit k serveru jiné verze (pouze dostanete varování)). Aktuální verze dokáží načíst dump z `pg_dump` verze 7.0.

Tip

Migraci můžeme zkrátit paralelním provozem nového serveru na jiném portu a migrací přes rouru:

```
pg_dumpall -p 5432 | psql -d postgres -p 6543
```

Ověřte si, že Vám nikdo v té době nepřistupuje k SQL serverům.

Postup při přechodu na novou verzi

Možné problémy

Migraci vždy testujte

- Ještě před dumpem v novější verzi vytvořete zálohu v aktuální verzi. Není zaručeno, že se starší klient dokáže připojit k novějšímu serveru, tj. bez této zálohy by cesta zpět mohla být obtížná.
- Prostudujte si odpovídající RELEASE NOTES.
- V ideálním případě máte k dispozici UNIT testy.
- Upgrade příliš neodkládejte, je jistější udělat několik menších kroků, než jeden skok. Nové verze PostgreSQL vycházejí jednou ročně. Zaručeně podporované jsou dvě verze zpátky (oprava chyb, bezpečnostní záplaty, atd). Jako optimální je upgrade každe dva roky.

Kde mohou nastat problémy?

- odstranění rizikových implicitních typových konverzí v 8.3, integrace fulltextu,
- přísnější kontrola UTF8 ve verzi 8.2 (tj. předchozí dump může být považován za nekorektní). Oprava - použití filtru `iconv`,
- příliš stará verze PostgreSQL (7.3 a starší) - provádějte upgrade inkrementálně.

Postup při přechodu na novou verzi

pg_upgrade

Tip

Pro upgrade větších databází je možné použít příkaz `pg_upgrade`. Je výrazně rychlejší než upgrade pomocí `pg_dump`, navíc s volbou `--link` nedochází ke kopírování souborů ***(Pozor: po startu nové verze už nikdy nelze nad stejnými daty nastartovat původní verzi)***.

Efektivní SQL

Chybné použití UNION

Pozn.

Je chybou používat UNION nad jednou tabulkou.

```
SELECT 'H', cena, cena*1.05
  FROM Tovar
 WHERE typ = 'sz' AND cena < 100
UNION
SELECT 'C', cena, cena*1.22
  FROM Tovar
 WHERE typ = 'dz' AND cena < 100
UNION
SELECT 'O', cena, cena*1.3
  FROM Tovar
 WHERE (cena >= 100) OR (typ <> 'dz')
```

Efektivní SQL

Správné použití CASE

Pozn.

Vzhledem k tomu, že CASE vrací record, je nutné jej rozvinout pomocí odvozené tabulky. ROW je anonymní konstruktor řádku. Poto je nutné přetypování.

```
CREATE TYPE nk AS (tp char(1), koef float);
SELECT (s.p).tp, s.cena, (s.p).koef*s.cena
  FROM
       (SELECT CASE WHEN typ = 'sz' AND cena < 100 THEN ROW('H',1.05)::nk
                    WHEN typ = 'dz' AND cena < 100 THEN ROW('C',1.22)::nk
                    ELSE ROW('O',1.33)::nk END, cena
          FROM Tovar
       ) AS s(p,cena);
```

Efektivní SQL

Nepoužívejte ALL

Pozn.

Při psaní vnořených dotazů lze chytře použít agregační funkce MIN a MAX. Následující dotazy jsou funkčně ekvivalentní. Druhý dotaz je ovšem řádově (1000x) rychlejší.

```
SELECT * FROM a
 WHERE a > ALL
       (SELECT b FROM b);

SELECT * FROM a
 WHERE a >
       (SELECT MAX(b) FROM b);
```

Efektivní SQL

Korelovaný dotaz v klauzuli SELECT nahraďte spojením s odvozenou tabulkou

Pozn.

Pro tabulky Produkt a Polozka (kardinalita 1:n) zobrazí 10 největších jednorázových prodejů.

```
SELECT *
  FROM
       (SELECT nazev,
                    (SELECT MAX(kusu)
                       FROM Polozka
                      WHERE produkt_id = p.id
                    ) AS kusu
          FROM Produkt p
       ) d
 WHERE d.kusu IS NOT NULL
 ORDER BY d.kusu DESC
 LIMIT 10
```

Efektivní SQL

Korelovaný dotaz v klauzuli SELECT nahraďte spojením s odvozenou tabulkou

Výhody

- čitelnější,
- nevyžaduje index na tab. položka(produkt_id),
- 2-3x rychlejší.

```
SELECT nazev, kusu
  FROM Produkt pr
       INNER JOIN
       (SELECT MAX(kusu) AS kusu, produkt_id
          FROM Polozka
         GROUP BY produkt_id
       ) AS po
       ON pr.id = po.produkt_id
 ORDER BY kusu DESC
 LIMIT 10
```

Efektivní SQL

Nicméně svět není jednoduchý

Pozn.

Zobrazí objednávky. Zdrojem jsou tabulky Zakaznik a Faktura.

```
SELECT z.nazev, f.vlozeno
  FROM Zakaznik z
       INNER JOIN
       Faktura f
       ON z.id = f.zakaznik_id
 WHERE f.vlozeno =
       (SELECT MAX(vlozeno)
          FROM Faktura
         WHERE zakaznik_id = z.id
       )
 ORDER BY f.vlozeno DESC
```

Efektivní SQL

Nicméně svět není jednoduchý

Pozn.

Tento dotaz je opět o něco čitelnější a cca 3x rychlejší. Stačí ale jediná změna.

```
SELECT z.nazev, f.vlozeno
  FROM Zakaznik z
       INNER JOIN
       (SELECT MAX(vlozeno) AS vlozeno, zakaznik_id
          FROM Faktura
         GROUP BY zakaznik_id
       ) f
       ON z.id = f.zakaznik_id
 ORDER BY f.vlozeno DESC
```

Efektivní SQL

Nicméně svět není jednoduchý

Pozn.

Zobrazí 20 nejnovějších objednávek. Zdrojem jsou tabulky Zakaznik a Faktura.

```
SELECT z.nazev, f.vlozeno
  FROM Zakaznik z
       INNER JOIN
       Faktura f
       ON z.id = f.zakaznik_id
 WHERE f.vlozeno =
       (SELECT MAX(vlozeno)
          FROM Faktura
         WHERE zakaznik_id = z.id
       )
 ORDER BY f.vlozeno DESC
 LIMIT 20
```

Efektivní SQL

Nicméně svět není jednoduchý

Pozn.

Po přidání **LIMIT 20** je korelovaný dotaz v klauzuli WHERE cca ***40x*** rychlejší.

```
SELECT z.nazev, f.vlozeno
  FROM Zakaznik z
       INNER JOIN
       (SELECT MAX(vlozeno) AS vlozeno, zakaznik_id
          FROM Faktura
         GROUP BY zakaznik_id
         ORDER BY vlozeno DESC
         LIMIT 20
       ) f
       ON z.id = f.zakaznik_id
 ORDER BY f.vlozeno DESC
```

Efektivní SQL

Nicméně svět není jednoduchý

Řešení

Změna zadání. 20 nejnovějších objednávek v měsíci prosinec 2009. Cílem je navrhnout GUI tak, aby poskytovalo dostatek indicií pro rychlé dotazy.

```
SELECT z.nazev, f.vlozeno
  FROM Zakaznik z
       INNER JOIN
       (SELECT MAX(vlozeno) AS vlozeno, zakaznik_id
          FROM Faktura
         WHERE vlozeno BETWEEN '2009-11-01' AND last_day('2009-11-01')
         GROUP BY zakaznik_id
         ORDER BY vlozeno DESC
         LIMIT 20
       ) f
       ON z.id = f.zakaznik_id
 ORDER BY f.vlozeno DESC
```

Indexy

Typy

Podporované typy

jednoduchý index jeden sloupec,
složený index více sloupců,
částečný index pouze nad částí tabulky,
funkční index nad výsledkem funkce.

Podporované formáty

- B-tree,
- Hash,
- GiST a GiN.

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] name ON table
    [ USING method ]
    ( { column | ( expression ) } [ opclass ] [, ...] )
    [ TABLESPACE tablespace ]
    [ WHERE predicate ]
```

Indexy

Ukázka funkčního a částečného indexu

Zadání

Urychlete vyhledávání uživatelů na základě zadání části doménové adresy, např. ”*.cvut.cz”. Pouze třetina uživatelů má emailovou adresu.

Pozn.

Index se v LIKE použije pouze tehdy pokud maska nezačíná žolíkem. Proto je nutné použít trik s revertováním adresy. cs_CZ locales si vynucuje použití opclass. Predikátem (částečný index) zvyšujeme selectivitu indexu. ***LIKE*** je case sensitive operace.

```
CREATE UNIQUE INDEX idx_users_email
  ON users((plvstr.rvrs(email)) varchar_pattern_ops)
 WHERE email IS NOT NULL
```

Indexy

Závěr

Doporučení

- indexy navrhujte pouze na sloupce s dostatečnou selectivitou,
- dobrými kandidáty na index jsou sloupce cizích klíčů a sloupce, pro agregační funkce MAX a MIN,
- nepoužívané indexy odstraňte,
- preferujte jednoduché indexy,
- indexy navrhujte pouze pro tabulky s více než 1000 řádky (nemají smysl),
- při změně charakteru dat vždy aktualizujte datové statistiky,
- cca jednou měsíčně (záleží na frekvenci změn v databázi) reindexujte databázi,
- v případě velkých tabulek zvažte přínos partitioningu.

Optimalizace dotazů

Pár rad

Doporučení

- optimalizujte časté nebo extrémně pomalé dotazy,
- před každou optimalizací nezapomeňte na *VACUUM ANALYZE*,
- predikáty zapisujte tak, aby jednu stranu zápisu tvořil pouze atribut,
- z výpisu prováděcího plánu určete smysl případných nových indexů,
- pokud se z nějakého důvodu nepoužije index, zkuste `set enable_seqscan to off`,
- pokud toto řešení pomůže zkuste zvětšit počet tříd datového histogramu sloupce indexu (má smysl u specifických rozdělení) `ALTER TABLE ALTER COLUMN SET STATISTICS`,
- přepište dotaz (EXISTS, IN, JOIN, ..).

Pozn.

Není chybou, když se nepoužije index v případě, že se jedná o malou tabulku nebo se z tabulky čte více než 30% řádků. Pokud si nejste jisti, penalizujte sekvenční čtení příkazem `set enable_seqscan to off`.

Optimalizace dotazů

Neoptimální plán, chybí index Faktura(zakaznik_id)

```
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Limit  (cost=74428635.52..74428635.54 rows=10 width=149)
   ->  Sort  (cost=74428635.52..74428636.30 rows=312 width=149)
         Sort Key: f.vlozeno
         ->  Nested Loop  (cost=0.00..74428622.59 rows=312 width=149)
               Join Filter: (f.vlozeno = (subplan))
               ->  Seq Scan on faktura f  (cost=0.00..1036.63 rows=62363 width=8)
               ->  Index Scan using zakaznik_pkey on zakaznik z
               (cost=0.00..0.11 rows=1 width=149)
                     Index Cond: (z.id = f.zakaznik_id)
               SubPlan
                 ->  Aggregate  (cost=1193.32..1193.33 rows=1 width=4)
                       ->  Seq Scan on faktura
                       (cost=0.00..1192.54 rows=312 width=4)
                             Filter: (zakaznik_id = $0)
(12 rows)
```

Optimalizace dotazů

Lepší plán, vytvořen index Faktura(zakaznik_id)

```
                                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------
 Limit  (cost=27081645.89..27081645.91 rows=10 width=149)
   ->  Sort  (cost=27081645.89..27081646.70 rows=324 width=149)
         Sort Key: f.vlozeno
         ->  Nested Loop  (cost=0.00..27081632.38 rows=324 width=149)
               Join Filter: (f.vlozeno = (subplan))
               ->  Seq Scan on faktura f  (cost=0.00..1060.23 rows=64723 width=8)
               ->  Index Scan using zakaznik_pkey on zakaznik z
               (cost=0.00..0.11 rows=1 width=149)
                     Index Cond: (z.id = f.zakaznik_id)
               SubPlan
                 ->  Aggregate  (cost=418.27..418.28 rows=1 width=4)
                       ->  Bitmap Heap Scan on faktura  (cost=6.44..417.46 rows=324 width=
                             Recheck Cond: (zakaznik_id = $0)
                             ->  Bitmap Index Scan on idx_faktura_zakaznik_id
                              (cost=0.00..6.44 rows=324 width=0)
                                   Index Cond: (zakaznik_id = $0)
(14 rows)
```

Optimalizace dotazů

Dobrý plán, přidán index Faktura(vlozeno)

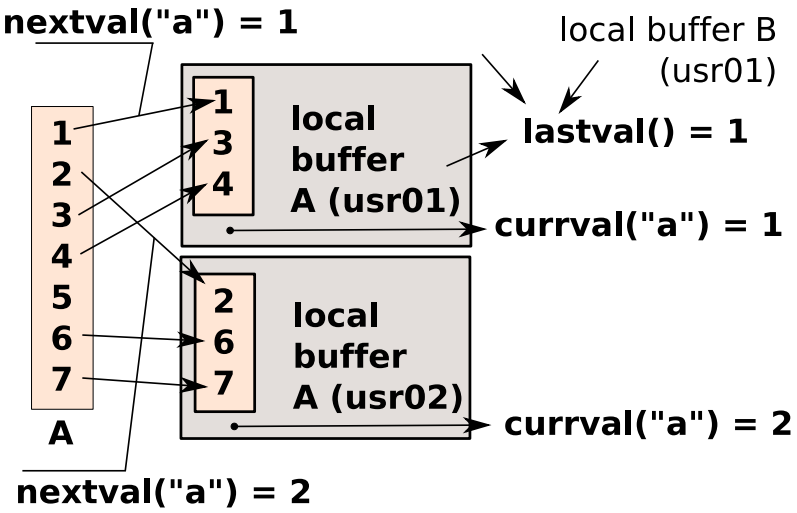
```
                                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.00..21086.84 rows=10 width=149)
   ->  Nested Loop  (cost=0.00..683213.69 rows=324 width=149)
         Join Filter: (f.vlozeno = (subplan))
         ->  Index Scan Backward using idx_faktura_vlozeno on faktura f
         (cost=0.00..3198.85 rows=64723 width=8)
         ->  Index Scan using zakaznik_pkey on zakaznik z (cost=0.00..0.11 rows=1 width=14
               Index Cond: (z.id = f.zakaznik_id)
         SubPlan
           ->  Result  (cost=10.37..10.38 rows=1 width=0)
                 InitPlan
                   ->  Limit  (cost=0.00..10.37 rows=1 width=4)
                         ->  Index Scan Backward using idx_faktura_vlozeno on faktura
                               (cost=0.00..3360.66 rows=324 width=4)
                             Filter: ((vlozeno IS NOT NULL) AND (zakaznik_id = $0))
(12 rows)
```

Sekvence

Schéma

Popis

- generují rostoucí posloupnost jedinečných čísel,
- používají se jako jedinečné kódy v rámci db nebo PK,
- ***víceuživatelsky bezpečné***.



Sekvence

Seznam funkcí

Funkce pro operace nad sekvencemi

- **nextval** zvýší aktuální hodnotu sekvence a tuto hodnotu vrátí, kromě toho zkopíruje tuto hodnotu do lokálního bufferu sekvence,
- **currval** vrací hodnotu uloženou v lokálním bufferu sekvence,
- **lastval** vrací hodnotu lokálního bufferu naposledy aktualizované sekvence,
- **setval** nastaví hodnotu sekvence.

Pozn.

Typ ***serial***, pro uživatele transparentně, automatizuje veškeré operace potřebné pro použití sekvencí v PK (analogie typu *identity* z jiných RDBMS).

Sekvence

Typ serial

```
postgres=> create table foo(pk serial primary key);
NOTICE:  CREATE TABLE will create implicit
sequence ‘‘foo_pk_seq’’ for serial column ‘‘foo.pk’’
NOTICE:  CREATE TABLE / PRIMARY KEY will create
implicit index ‘‘foo_pkey’’ for table ‘‘foo’’
CREATE TABLE
postgres=> \d foo
                        Table ‘‘public.foo’’
 Column |  Type   |                    Modifiers
--------+---------+-------------------------------------------------
 pk     | integer | not null default nextval('foo_pk_seq'::regclass)
Indexes:
    ‘‘foo_pkey’’ PRIMARY KEY, btree (pk)
```

Integrovaný fulltext

Instalace

```
-- Soubory czech.affix, czech.dict a czech.stop rozbalte
-- a přesuňte do adresáře share/tsearch_data.
CREATE TEXT SEARCH DICTIONARY cspell
   (template=ispell, dictfile = czech, afffile=czech,
    stopwords=czech);

CREATE TEXT SEARCH CONFIGURATION cs (copy=english);

ALTER TEXT SEARCH CONFIGURATION cs
   ALTER MAPPING FOR word, asciiword WITH cspell, simple;
```

Integrovaný fulltext

Verifikace

```
postgres=# select * from ts_debug('cs','Příliš žluťoučký kůň se napil žluté vody');
   alias   |    description    |   token   |  dictionaries   | dictionary |   lexemes
-----------+-------------------+-----------+-----------------+------------+-------------
 word      | Word, all letters | Příliš    | {cspell,simple} | cspell     | {příliš}
 blank     | Space symbols     |           | {}              |            |
 word      | Word, all letters | žluťoučký | {cspell,simple} | cspell     | {žluťoučký}
 blank     | Space symbols     |           | {}              |            |
 word      | Word, all letters | kůň       | {cspell,simple} | cspell     | {kůň}
 blank     | Space symbols     |           | {}              |            |
 asciiword | Word, all ASCII   | se        | {cspell,simple} | cspell     | {}
 blank     | Space symbols     |           | {}              |            |
 asciiword | Word, all ASCII   | napil     | {cspell,simple} | cspell     | {napít}
 blank     | Space symbols     |           | {}              |            |
 word      | Word, all letters | žluté     | {cspell,simple} | cspell     | {žlutý}
 blank     | Space symbols     |           | {}              |            |
 asciiword | Word, all ASCII   | vody      | {cspell,simple} | cspell     | {voda}
(13 rows)
```

Integrovaný fulltext

Použití - vytvoření fulltextového indexu

```
CREATE TABLE fulltext_test(zdroj text, nazev text);

\set content '\'' `sed -e ``s/'/\\\\\\'/g'' < ~/postgres/dotazy.html` '\''

INSERT INTO fulltext_test VALUES(:content, 'Korelované poddotazy');

CREATE INDEX pgweb_idx ON fulltext_test
   USING gin(to_tsvector('cs', zdroj));
```

Integrovaný fulltext

Použití - použití fulltextového vyhledávání

```
postgres=# SELECT nazev, ts_headline('cs', zdroj, to_tsquery('cs', 'efektivní & dotaz'))

             FROM fulltext_test
            WHERE to_tsvector('cs',zdroj) @@ to_tsquery('cs','efektivní & dotaz');
                                                        ts_headline

        nazev         |                                                                  ts_headli
----------------------+-------------------------------------------------------------------------------
 Korelované poddotazy | <b>dotazy</b> <b>efektivně</b> nahradit. Pozn. v řadě aplikací jsou
 Left inner join      | <b>dotaz</b>
                      : vyžaduje všechny sloupce tabulky, pak je to v pořádku. To však v pří
                      : většinou nebývá tak úplně pravda, a tudíž zpracování příkazu není ta
(2 rows)
```

Integrovaný fulltext

Vyhledávání na základě prefixu

Popis

Fulltext umožňuje specifikovat hledaná slova prefixem:

```
postgres=# select *
             from codebooks.psc_obce
            where to_tsvector('simple', cast_obce)
                    @@ to_tsquery('simple', 'Bene:*');
         obec          | psc   |      nazev_posty      | lau1
-----------------------+-------+-----------------------+-------
 Benecko               | 51237 | Benecko               | CZ0514
 Benecko               | 51401 | Jilemnice             | CZ0514
 Bušanovice            | 38422 | Vlachovo Březí        | CZ0315
 Broumov               | 55001 | Broumov 1             | CZ0523
 Benešov               | 25601 | Benešov u Prahy       | CZ0201
 Benešov               | 67953 | Benešov u Boskovic    | CZ0641
```

Vyhledávání na základě podobnosti

Modul pg_tgrm

Popis

Vyhledávání na základě částečné shody nelze urychlit indexem. Poměrně úspěšně lze použít vyhledávání pomocí podobnosti trigramů řetězců.

- Výhoda: lze indexovat,
- Nevýhoda, nutno nastavovat limit (čím delší řetězec, tim víc možných trigramů, tím vyšší míra podobnosti, tím vyšší nezbytný limit.

V následujícím příkladu používám trigram vyhledávání pro urychlení vyhledání v číselníku obcí - hledám PSČ.

```
postgres=# SELECT show_trgm('Benesov');
                  show_trgm
-------------------------------------------
 {"  b","  be",ben,ene,eso,nes,"ov ",sov}

postgres=# CREATE INDEX trgm ON psc
               USING gin(replace(obec,' ','') gin_trgm_ops);
CREATE INDEX
```

Vyhledávání na základě podobnosti

Modul pg_tgrm

```
postgres=# SELECT * FROM psc WHERE replace(obec,' ','') % 'benesov';
         obec          | okres | kraj | pscn
-----------------------+-------+------+-------
 Kozmice u Benesova    | 01093 | 01   | 25725
 Bystrice u Benesova   | 01083 | 01   | 25751
 Benesov u Prahy       | 01064 | 01   | 25601
 Benesov nad Cernou    | 02027 | 02   | 38282
 Bezverov              | 03016 | 03   | 33041
 Unesov                | 03016 | 03   | 33038

postgres=# EXPLAIN ANALYZE SELECT * FROM psc WHERE replace(obec,' ','') % 'benesov';
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on psc  (cost=4.29..17.28 rows=4 width=30)
 (actual time=2.236..3.227 rows=11 loops=1)
   Preread target: 64
   Filter: (replace((obec)::text, ' '::text, ''::text) % 'benesov'::text)
   ->  Bitmap Index Scan on trgm  (cost=0.00..4.29 rows=4 width=0)
       (actual time=1.866..1.866 rows=130 loops=1)
         Index Cond: (replace((obec)::text, ' '::text, ''::text) % 'benesov'::text)
 Total runtime: 3.384 ms
(6 rows)
```

Vyhledávání na základě podobnosti

Modul pg_tgrm

```
postgres=# EXPLAIN ANALYZE select * FROM psc WHERE (obec ilike '%benesov%');
                                              QUERY PLAN
-----------------------------------------------------------------------------------------------
 Seq Scan on psc  (cost=0.00..109.40 rows=1 width=30)
 (actual time=1.083..20.058 rows=9 loops=1)
   Filter: ((obec)::text ~~* '%benesov%'::text)
 Total runtime: 20.146 ms
(3 rows)

postgres=# SELECT * FROM psc WHERE (obec ILIKE '%benesov%');
         obec          | okres | kraj | pscn
-----------------------+-------+------+-------
 Kozmice u Benesova    | 01093 | 01   | 25725
 Bystrice u Benesova   | 01083 | 01   | 25751
 Benesov u Prahy       | 01064 | 01   | 25601
 Benesov nad Cernou    | 02027 | 02   | 38282
 Benesov nad Ploucnici | 04011 | 04   | 40722
 Benesov u Boskovic    | 06079 | 06   | 67953
 Dolni Benesov         | 07061 | 07   | 74722
 Horni Benesov         | 09044 | 09   | 79312
 Benesov u Semil       | 11005 | 11   | 51206
(9 rows)
```

Vyhledávání na základě podobnosti

Modul pg_trgm

```
postgres=# PREPARE filter(varchar) AS
              SELECT *
                 FROM psc
                WHERE replace(obec,' ','') % $1
                  AND obec ilike '%'||$1||'%';
PREPARE
postgres=# EXECUTE filter('benesov');
         obec           | okres | kraj | pscn
------------------------+-------+------+-------
 Kozmice u Benesova     | 01093 | 01   | 25725
 Bystrice u Benesova    | 01083 | 01   | 25751
 Benesov u Prahy        | 01064 | 01   | 25601
 Benesov nad Cernou     | 02027 | 02   | 38282
 Benesov nad Ploucnici  | 04011 | 04   | 40722
 Benesov u Boskovic     | 06079 | 06   | 67953
 Dolni Benesov          | 07061 | 07   | 74722
 Horni Benesov          | 09044 | 09   | 79312
 Benesov u Semil        | 11005 | 11   | 51206
(9 rows)

postgres=# EXPLAIN ANALYZE EXECUTE filter('benesov');
                                                              QUERY PLAN
----------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on psc  (cost=4.29..17.31 rows=1 width=30) (actual time=4.241..7.319 rows=9 loops=1)
   Preread target: 64
   Filter: ((replace((obec)::text, ' '::text, ''::text) % ($1)::text) AND ((obec)::text ~~* (('%'::text || ($1)::text) |
   ->  Bitmap Index Scan on trgm  (cost=0.00..4.29 rows=4 width=0)
   (actual time=4.064..4.064 rows=130 loops=1)
         Index Cond: (replace((obec)::text, ' '::text, ''::text) % ($1)::text)
 Total runtime: 7.494 ms
(6 rows)
```

Pole

Úvod

Popis

- jasně porušují první normální formu (ale je atom skutečně nedělitelný),
- praktický typ pro ukládání časových řad, nezastupitelný v plpgsql,
- všechny prvky pole musí být jednoho typu,
- pole může obsahovat *NULL*.

```
postgres=# select ARRAY['Pavel','Tomáš'];
      array
-----------------
 {Pavel,Tomáš}
(1 row)

postgres=# select '{Pavel, Tomáš}'::varchar[];
           varchar
------------------------------
 {Pavel, Tomáš}::varchar[];
```

Pole

Operace, rozvinutí pole

```
postgres=> CREATE OR REPLACE FUNCTION unnest(anyarray)
postgres-> RETURNS SETOF anyelement AS $$
postgres$$> SELECT $1[idx.i]
postgres$>   FROM generate_series(array_lower($1,1), array_upper($1,1))
postgres$>        AS idx(i) $$
postgres-> LANGUAGE sql IMMUTABLE;
CREATE FUNCTION
postgres=> SELECT unnest(ARRAY[1,2,3]);
 unnest
--------
      1
      2
      3
(3 rows)

CREATE OR REPLACE FUNCTION unnest(anyarray)
RETURNS SETOF anyelement AS $$
  SELECT $1[idx.i]
     FROM generate_subscripts($1,1) AS idx(i)
$$ LANGUAGE sql IMMUTABLE;
```

Pole

Operace, sestavení pole a rozšiřování pole

```
postgres=# SELECT ARRAY
            (SELECT *
               FROM (VALUES('Pavel'),('Tomáš')) a
            ) as output;
     output
-----------------
 {Pavel,Tomáš}
(1 row)

postgres=# SELECT ARRAY['a','b','c'] || text 'd';
 ?column?
-----------
 {a,b,c,d}
(1 row)

postgres=# SELECT ARRAY['a','b','c'] || ARRAY['d','e'];
  ?column?
-------------
 {a,b,c,d,e}
(1 row)
```

Pole

Transformace pole na relaci a relaci na pole

```
postgres=# SELECT array_agg(x) FROM (VALUES(10),(20),(30)) g(x);
 array_agg
------------
 {10,20,30}
(1 row)

postgres=# SELECT unnest(ARRAY[10,20,30]);
 unnest
--------
     10
     20
     30
(3 rows)
```

Pole

Operace, vyhledávání I.

Seznam operátorů

- `@>` obsahuje,
- `@<` je obsaženo,
- `&&` průnik.
- `=` rovnost,
- `<>` nerovnost.

```
postgres=#
CREATE OR REPLACE FUNCTION ra()
RETURNS int[] AS $$
  SELECT ARRAY
              (SELECT round(random()*10)::int
                 FROM generate_series(1, round(random()*10)::int)
              )::int[];
$$$ LANGUAGE sql VOLATILE;
CREATE FUNCTION
```

Pole

Operace, vyhledávání II.

```
postgres=# CREATE TABLE Omega(a int[]);
postgres=# CREATE INDEX idx_omega_a ON Omega USING GIN (a);

postgres=# INSERT INTO Omega
postgres-#   SELECT ra() FROM generate_series(1,100000);
INSERT 0 100000

postgres=# \timing
Timing is on.
postgres=# SELECT *
             FROM omega
            WHERE a @> array[1,2,3,4,5,6,7,8,9,10];
           a
-----------------------
 {6,4,2,3,8,10,1,7,5,9}
 {9,6,3,1,7,2,8,5,10,4}
(2 rows)

Time: 85,386 ms
```

Pole

Operace, vyhledávání III.

Modul intarray, GiST index

- sestavení indexu je několikanásobně pomalejší než GiN indexu,
- při hledání pole o malém počtu pomalejší než GiN index,
- s větším počtem vyhledávaných prvků (menší výsledná množina) roste rychlost (GiN se chová opačně).

```
postgres=# CREATE INDEX idx_omega_a_gist
             ON Omega USING GIST(a gist__int_ops);

postgres=# SELECT *
             FROM Omega WHERE a @> array[1,2,3,4,5,6,7,8,9,10];
           a
-----------------------
 {6,4,2,3,8,10,1,7,5,9}
 {9,6,3,1,7,2,8,5,10,4}
(2 rows)

Time: 36,071 ms
```

Funkce generate_series

Cyklus v SQL

Popis

- generuje jednosloupcovou tabulku s číselnou posloupností,
- umožňuje snadné generování testovacích dat a benchmarků,
- umožňuje realizovat funkci FOR v čistém SQL.

```
FUNCTION generate_series(integer, integer [, integer])
RETURNS SETOF integer

postgres=# SELECT idx.i
             FROM generate_series(1,2) AS idx(i);
 i
---
 1
 2
(2 rows)
```

Funkce generate_series

Příklad

Popis

Funkce *rvrs* provede prohození pořadí znaků v řetězci.

```
postgres=# CREATE OR REPLACE FUNCTION rvrs(varchar)
postgres-# RETURNS varchar AS $$
postgres$$#   SELECT array_to_string(ARRAY
postgres$#          (SELECT substring($1, s.i, 1)
postgres$#             FROM generate_series(length($1),1,-1) AS s(i)
postgres$#          ), '');
postgres$# $$ LANGUAGE SQL;
CREATE FUNCTION
postgres=# SELECT rvrs('abcde');
 rvrs
-------
 edcba
(1 row)
```

Školení PostgreSQL efektivně

Všeobecná část + administrace - podklady

Pavel Stěhule

http://www.postgres.cz

16. 10. 2023