Useful (yet frequently omitted) extensions

Tomas Vondra, GoodData

tomas.vondra@gooddata.com / tomas@pgaddict.com
42
contrib

- 44 (42) modules included in PostgreSQL
  http://www.postgresql.org/docs/devel/static/static/contrib.html

- examples of extensibility
  - new data types, index support, FDW, …

- administration tools
  - monitoring of queries, various analysis tools, …

- libraries of useful functions
  - pgcrypto, adminpack, …
adminpack auth_delay auto_explain btree_gin btree_gist chkpass citext cube dblink dict_int dict_xsyn dummy_seclabel earthdistance file_fdw fuzzystromatch hstore intagg intarray isn lo ltree pageinspect passwordcheck pg_buffercache pgcrypto pg_freespacemap pg_prewarm pgrowlocks pg_stat_statements pgstattuple pg_trgm postgres_fdw seg sepgsql spi sslinfo tablefunc tcn test_parser test_shm_mq tsearch2 unaccent uuid-ossp xml2
adminpack auth_delay auto_explain
**btree_gin** **btree_gist** chkpass citext cube
dblink dict_int dict_xsyn dummy_seclabel
earthdistance file_fdw fuzzystrmatch
hstore intagg intarray isn lo ltree
pageinspect passwordcheck pg_buffercache
pgcrypto pg_freespacemap pg_prewarm
pgrowlocks pg_stat_statements
pgstattuple pg_trgm postgres_fdw seg
sepgsql spi sslinfo tablefunc tcn
test_parser test_shm_mq tsearch2
unaccent uuid-ossp xml2
CREATE TABLE products (  
id SERIAL PRIMARY KEY,  
category_id INTEGER,  
description TEXT,  
fulltext TSVECTOR,  
...  
);

CREATE INDEX product_fts_idx ON products USING GIST (category_id, fulltext);

SELECT * FROM products  
WHERE category_id = 12345  
AND fulltext @@ to_tsquery('hello');
CREATE TABLE products (  
id SERIAL PRIMARY KEY,  
category_id INTEGER,  
description TEXT,  
fulltext TSVECTOR,  
...)
);

CREATE INDEX product_fts_idx ON products  
USING GIST (category_id, fulltext);

ERROR: data type integer has no default operator class for access method "gist"

:- (}
• **B-tree** - regular (tree-like) indexes
  - standard "scalar" data types (INT, TEXT, ...)
  - <, <=, =, >=, >

• **GIN / GiST** - "space" indexes
  - “vector” types (tsvector, arrays, ranges)
  - <<, &<, &>, >>, <<-|, &<|, |&>, |>>>, @>, <@, ~=, &&
  - overlaps, same, contains, ...

• **btree_gin / btree_gist**
  - GIN / GiST operator classes for scalar data types
  - emulation based on btree opclass
CREATE TABLE products ( 
    id SERIAL, 
    category_id INTEGER, 
    description TEXT, 
    fulltext TSVECTOR, 
    ...
); 

CREATE EXTENSION btree_gist; 

CREATE INDEX product_fts_idx ON products 
    USING GIST (category_id, fulltext); 

;) :-)
Why not to use two standalone indexes?
  and then combine using bitmap index scan(s)

EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 100
  AND unique2 > 9000;

QUERY PLAN
----------------------------------------------------------
Bitmap Heap Scan on tenk1  (cost=11.27..49.11 rows=11 width=244)
  Recheck Cond: ((unique1 < 100) AND (unique2 > 9000))
  ->  BitmapAnd  (cost=11.27..11.27 rows=11 width=0)
    ->  Bitmap Index Scan on tenk1_unique1  (cost= ... Index Cond: (unique1 < 100)
    ->  Bitmap Index Scan on tenk1_unique2  (cost= ... Index Cond: (unique2 > 9000)
Why not to use two standalone indexes?
  ○ and then combine using bitmap index scan(s)

EXPLAIN SELECT * FROM tenk1 WHERE unique1 < 100
  AND unique2 > 9000;

QUERY PLAN

----------------------------------------------------------
Bitmap Heap Scan on tenk1  (cost=11.27..49.11 rows=11 width=244)
  Recheck Cond: ((unique1 < 100) AND (unique2 > 9000))
  ->  BitmapAnd  (cost=11.27..11.27 rows=11 width=0)
     ->  Bitmap Index Scan on tenk1_unique1  (cost= ...  
         Index Cond: (unique1 < 100)
     ->  Bitmap Index Scan on tenk1_unique2  (cost= ...  
         Index Cond: (unique2 > 9000)

but sometimes a single index is required ...
● exclusion constraints
  ○ a constraint resembling UNIQUE
  ○ e.g. with ranges we require that they don’t overlap

● example - system for booking meeting rooms
  ○ reservations for the same room must not overlap

```
CREATE TABLE room_bookings (  
id SERIAL PRIMARY KEY,  
room_id INT NOT NULL REFERENCES ...,
valid_from TIMESTAMP,  
valid_to TIMESTAMP  
);
```
• exclusion constraints
  ○ a constraint resembling UNIQUE
  ○ e.g. with ranges we require that they don’t overlap

• example - system for booking meeting rooms
  ○ reservations for the same room must not overlap

CREATE TABLE room_bookings (  
  id SERIAL PRIMARY KEY,  
  room_id INT NOT NULL REFERENCES ... ,  
  valid_from TIMESTAMP,  
  valid_to TIMESTAMP,  
  EXCLUDE USING gist (room_id WITH =,  
      TSRANGE(valid_from, valid_to) WITH &&)  
);
• exclusion constraints
  ○ a constraint resembling UNIQUE
  ○ e.g. with ranges we require that they don’t overlap

• example - system for booking meeting rooms
  ○ reservations for the same room must not overlap

CREATE TABLE room_bookings (  
id SERIAL PRIMARY KEY,  
room_id INT NOT NULL REFERENCES ...,  
validity TSRANGE,  
EXCLUDE USING gist (room_id WITH =,  
  validity WITH WITH &&)  
);
adminpack auth_delay auto_explain
btree_gin btree_gist chkpass citext cube
dblink dict_int dict_xsyn dummy_seclabel
earthdistance file_fdw fuzzystrmatch
hstore intagg intarray isn lo ltree
pageinspect passwordcheck pg_buffercache
pgcrypto pg_freespacemap pg_prewarm
pgrowlocks pg_stat_statements
pgstattuple pg_trgm postgres_fdw seg
sepgsql spi sslinfo tablefunc tcn
test_parser test_shm_mq tsearch2
unaccent uuid-ossp xml2
adminpack auth_delay auto_explain
btree_gin btree_gist chkpass citext cube
dblink dict_int dict_xsyn dummy_seclabel
earthdistance file_fdw fuzzystrmatch
hstore intagg intarray isn lo ltree
pageinspect passwordcheck pg_buffercache
pgcrypto pg_freespacemap pg_prewarm
pgrowlocks pg_stat_statements
pgstattuple pg_trgm postgres_fdw seg
sepgsql spi sslinfo tablefunc tcn
test_parser test_shm_mq tsearch2
unaccent uuid-ossp xml2
CREATE TABLE inventory_csv (
    product_id   INT PRIMARY KEY,
    num_of_pcs   INT NOT NULL
);
CREATE TABLE inventory_csv (  
  product_id  INT  PRIMARY KEY,  
  num_of_pcs  INT  NOT NULL  
);

COPY inventory_csv FROM '/data/inventory.csv'  
  WITH (format csv);
CREATE TABLE inventory_csv (  
    product_id INT PRIMARY KEY,  
    num_of_pcs INT NOT NULL  
);

COPY inventory_csv FROM '/data/inventory.csv'  
    WITH (format csv);

SELECT * FROM inventory_csv;

-- MERGE ~ (INSERT + UPDATE)
CSV file \(\xrightarrow{MERGE}\) inventory table
CREATE EXTENSION file_fdw;

CREATE SERVER file_server
FOREIGN DATA WRAPPER file_fdw;

CREATE FOREIGN TABLE inventory_fdw (product_id INT,
num_of_pcs INT)
SERVER file_server OPTIONS (filename '/data/inventory.csv', format 'csv');
WITH updated_ids as ( 
    UPDATE inventory 
    SET num_of_items = src.num_of_items 
    FROM inventory_fdw src 
    WHERE inventory.product_id = src.product_id 
    RETURNING inventory.product_id 
)

INSERT INTO inventory 
SELECT product_id, num_of_items 
FROM inventory_fdw 
WHERE product_id NOT IN (SELECT * FROM updated_ids);

- usual way to do this
  - read data into a temporary table using COPY
  - use the same writable CTE
adminpack auth_delay auto_explain
btree_gin btree_gist chkpass citext cube
dblink dict_int dict_xsyn dummy_seclabel
earthdistance file_fdw fuzzystrmatch
hstore intagg intarray isn lo ltree
pageinspect passwordcheck pg_buffercache
pgcrypto pg_freespacemap pg_prewarm
pgrowlocks pg_stat_statements
pgstattuple pg_trgm postgres_fdw seg
sepgsql spi sslinfo tablefunc tcn
test_parser test_shm_mq tsearch2
unaccent uuid-ossp xml2
CREATE EXTENSION hstore;

SELECT 'a=>1, b=>2, c=>3'::hstore;

   hstore
---------------------
   "a":"1", "b":"2", "c":"3"

<table>
<thead>
<tr>
<th>id (int)</th>
<th>full_name  (text)</th>
<th>custom_info (hstore)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alice Cooper</td>
<td>born =&gt; 1948, country =&gt; USA, state =&gt; Michigan, genre =&gt; shock rock, name =&gt; Vincent Damon Furnier</td>
</tr>
<tr>
<td>2</td>
<td>Ozzy Osbourne</td>
<td>born =&gt; 1948, country =&gt; england, genre =&gt; rock, hobby =&gt; drugs, favourite_meal =&gt; bats, nick =&gt; Prince of darkness</td>
</tr>
<tr>
<td>3</td>
<td>Justin Bieber</td>
<td>country =&gt; canada, genre =&gt; crap, born =&gt; 1994</td>
</tr>
<tr>
<td>4</td>
<td>Bryan Adams</td>
<td>country =&gt; canada, born =&gt; 1959, genre =&gt; unknown</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
hstore ? key
   does the hstore contain key "key"?

hstore -> key
   returns value for key "key"

hstore @$>$ hstore
   is the second hstore contained in the first one (all keys with exactly the same values)?

hstore - key
   delete key from a hstore (subset)

hstore - hstore
   delete subset from a hstore (matching pairs)

And much more, check the docs!
we want to store e-mail archive
  ○ we want to keep all of them (including headers)
  ○ and we need to query them easily

CREATE TABLE messages (
  id INT PRIMARY KEY,
  body TEXT,
  date_sent TIMESTAMP,
  addr_from TEXT,
  addr_to TEXT[],
  addr_cc TEXT[],
  subject TEXT,
  ... and ~million of other headers ... 
);
● a separate column for each possible header
  ○ explained on the previous slide
  ○ ultra-ultra-wide tables (sparsely filled)
  ○ ... and futile thanks to custom headers :-(

● EAV schema
  ○ better, used quite commonly
  ○ inefficient with multi-header queries (multiple joins)

<table>
<thead>
<tr>
<th>messages</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INT (PK)</td>
<td></td>
</tr>
<tr>
<td>sent</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>from</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>subject</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>body</td>
<td>TEXT</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>headers</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>INT (PK)</td>
<td></td>
</tr>
<tr>
<td>message_id</td>
<td>INT (FK)</td>
<td></td>
</tr>
<tr>
<td>header_name</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>header_value</td>
<td>TEXT</td>
<td></td>
</tr>
</tbody>
</table>
hstore to the rescue!

CREATE TABLE messages (  
id INT PRIMARY KEY,  
...  
reply_to TEXT,  
subject TEXT,  
body TEXT,  
headers HSTORE
);

CREATE INDEX messages_headers_idx  
ON messages USING GIST (headers);

SELECT * FROM messages  
WHERE headers ? 'x-spam-flag';

SELECT (headers -> "content-type") FROM messages  
WHERE headers @> ""message-id" => "<eWswd@gmail.com>"";
jsonb

- hopefully will get into 9.4 (core)
- JSON stored in a binary format
  - do not confuse with BSON
- several benefits over hstore
  - de facto standard format for this kind of data
  - hierarchical (tree-ish, nested)
  - data types (JSON)
- improved GIN indexing
adminpack auth_delay auto_explain btree_gin btree_gist chkpass citext cube
dblink dict_int dict_xsyn dummy_seclabel
earthdistance file_fdw fuzzystrmatch hstore intagg intarray isn lo ltree
pageinspect passwordcheck pg_buffercache pgcrypto pg_freespacemap pg_prewarm
pgrowlocks pg_stat_statements pgstattuple pg_trgm postgres_fdw seg
sepgsql spi sslinfo tablefunc tcn
test_parser test_shm_mq tsearch2 unaccent uuid-ossp xml2
- tree-like structure in a relational DB is a PITA ;-)
- e.g. a hierarchy of categories in an e-shop

```sql
CREATE TABLE categories (  
id         INT PRIMARY KEY,  
parent_id  INT REFERENCES categories(id),  
title      TEXT  
);
```

- ... now try to search (for a given category)
  - all parent categories (path to root)
  - all (not just direct) subcategories
  - all the products (including subcategories)

- not quite effective :-(
- **LTREE data type - path through a tree**

Components
Components.GPU
Components.RAM
Computers
Computers.Desktops
Computers.Laptops

```sql
CREATE TABLE categories (
    id         INT PRIMARY KEY,
    path       LTREE UNIQUE,
    title      TEXT
);
```
- cycles not possible (unlike with the FK)
- there can be "gaps" (like with the FK)

rich querying options

- ltree vs. ltree

```
SELECT * FROM categories WHERE 'Computers' @> path;
```

- ltree vs. lquery - simple queries

```
SELECT * FROM categories WHERE path ~ '*.Computers.*';
```

- ltree vs. ltxtquery - fulltext queries (AND, OR, ...)

```
SELECT * FROM categories WHERE path @ 'GPU & RAM';
```
adminpack auth_delay auto_explain
btree_gin btree_gist chkpass citext cube
dblink dict_int dict_xsyn dummy_seclabel
earthdistance file_fdw fuzzystrimatch
hstore intagg intarray isn lo ltree
pageinspect passwordcheck pg_buffercache
pgcrypto pg_freespacemap pg_prewarm
pgrowlocks pg_stat_statements
pgstattuple pg_trgm postgres_fdw seg
sepgsql spi sslinfo tablefunc tcn
test_parser test_shm_mq tsearch2
unaccent uuid-ossp xml2
• **trigrams** - splitting the text into three-letter groups

```sql
SELECT show_trgm('car');
```

```
show_trgm
---------------------
{"  c"," ca","ar ",car}
```

• **word similarity**

```sql
SELECT similarity('cars', 'carrots');
```

```
similarity
----------
    0.3
```
• use case - typo correction (search from UI ...)

```sql
CREATE TABLE dict (word text);

... fill it with english dictionary ...

SELECT word FROM dict
     ORDER BY similarity(‘somethinc’, word) DESC
     LIMIT 10;
```

• the query returns 10 most ‘similar’ words

• but it’s going to be rather slow (sequential scans)
  ○ especially if you use large dictionary
  ○ thus not really useful in UI
• similarity is ~ inverse concept to distance

```
SELECT ('cars' <-> 'carrots') AS distance;
```

```
distance          = (1 - similarity)
----------         -------------------
0.7          =        (1 - 0.3)
```

• and distance is the basis of GIN/GiST indexes

```
CREATE INDEX trgm_word_idx ON dict
    USING GIST (word gist_trgm_ops);
```

```
SELECT word, (words <-> 'search') AS distance
    FROM dict ORDER BY distance DESC LIMIT 10;
```

```
SELECT word FROM dict WHERE word LIKE '%aaa%';  -- 9.1
SELECT word FROM dict WHERE word ~ '(aaa|bbb)'; -- 9.3
```
adminpack auth_delay auto_explain btree_gin btree_gist chkpass citext cube dblink dict_int dict_xsyn dummy_seclabel earthdistance file_fdw fuzzystarmatch hstore intagg intarray isn lo ltree pageinspect passwordcheck pg_buffercache pgcrypto pg_freespacemap pg_prewarm pgrowlocks pg_stat_statements pgstattuple **pg_trgm** postgres_fdw seg sepgsql spi sslinfo tablefunc tcn test_parser test_shm_mq tsearch2 unaccent uuid-ossp xml2
pgxn.org

- independent / inofficial extension repository
- website with searching etc.
- tools for easier publishing / installing
  - pgxn client
  - fetch / extract / install into a database

$ sudo apt-get install pgxnclient
$ pgxnclient --help
$ pgxnclient install quantile
$ pgxnclient load -d testdb quantile
quantile

- written by me
- computes percentiles (yeah, wrong name)

```sql
SELECT
department_id,
    `quantile`(salary, 0.5) AS salary_median,
    `quantile`(salary, [0.25, 0.5, 0.75]) AS quartiles,
FROM employees
GROUP BY department_id;
```
pg_repack

- traditional maintenance of tables / indexes

  VACUUM FULL huge_and_important_table;

  CLUSTER huge_and_important_table;
pg_repack

- traditional maintenance of tables / indexes
  
  VACUUM FULL huge_and_important_table;

  CLUSTER huge_and_important_table;

- usually to get rid of (index) bloat
  - e.g. after removing “old” data

- … lots of screams from users / boss / …
pg_repack

• with pg_repack

  -- VACUUM FULL
  $ pg_repack --no-order \n      --table huge_and_important_table mydb

  -- CLUSTER
  $ pg_repack --table huge_and_important_table mydb

• can’t handle DDL (data corruption)
  ○ the reason why not in the core (yet)
pg_partman

- partition management
  - suite of PL/pgSQL function + python scripts (admin)
  - two variants - by ID or time
  - custom granularity and retention

- inheriting parameters from the parent table
  - default values, indexes, constraints
  - access rights / ownership

- great if it matches your use-case
• so let’s create a partitioned table

CREATE TABLE my_parent_table (  
    col1 SERIAL PRIMARY KEY,  
    col2 TEXT,  
    col3 TIMESTAMPTZ DEFAULT now()  
);

SELECT part.create_parent('my_parent_table',  
                          'col3', 'time-static', 'daily');

• create new table regularly (from cron)

0 0 * * * psql mydb -c "run_maintenance()"
... and many others

- **pgTAP**
  - unit testing for TAP (Test Anything Protocol)
- **plproxy**
  - a procedural language implementing sharding
- **plv8**
  - JavaScript as a procedural language (V8 engine)
- **s3_fdw**
  - FDW access to S3 on the Amazon WS
- **semver**
  - data type for semantic versioning (semver.org)
Questions?