PostgreSQL Extensions
Advanced Use Cases

Dimitri Fontaine dimitri@2ndQuadrant.fr

January 31, 2014
• pgloader, prefix, skytools, ...
• apt.postgresql.org
• CREATE EXTENSION
• CREATE EVENT TRIGGER
• MySQL migration tool, new pgloader version
Advanced Extension Use Cases

Agenda

- How PostgreSQL extensibility works
- Things you can do with a PostgreSQL Extension
- The PostgreSQL indexing Framework
- How to solve some practical use cases with existing extensions
PostgreSQL is highly extensible
select col1, col2 from table where col1 = 'something';
SELECT col
  FROM table
WHERE stamped > date 'today' - interval '1 day';
select iprange, locid
from geolite.blocks
where iprange >>= '91.121.37.122';

<table>
<thead>
<tr>
<th>iprange</th>
<th>locid</th>
</tr>
</thead>
<tbody>
<tr>
<td>91.121.0.0-91.121.159.255</td>
<td>75</td>
</tr>
</tbody>
</table>

(1 row)

Time: 1.220 ms
SQL Operators are all dynamic and found in the catalogs.

```sql
select amopopr::regoperator
from pg_opclass c
join pg_am am
  on am.oid = c.opcmethod
join pg_amop amop
  on amop.amopfamily = c.opcfamily
where opcintype = 'ip4r'::regtype
and am.amname = 'gist';
```

```
<table>
<thead>
<tr>
<th>amopopr</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;=</td>
</tr>
<tr>
<td>&lt;&lt;=</td>
</tr>
<tr>
<td>&gt;&gt;=</td>
</tr>
<tr>
<td>&lt;&lt;=</td>
</tr>
<tr>
<td>&amp;&amp;</td>
</tr>
<tr>
<td>=</td>
</tr>
</tbody>
</table>

(6 rows)
```
select id, name, pos,
    pos <@> point(-6.25, 53.34) as miles
from pubnames
order by pos <-> point(-6.25, 53.34)
limit 10;
PostgreSQL is Extensible

PostgreSQL plugins are data types and index support

• Data Type
• Input/Output functions
• Casts
• Operator Classes
PostgreSQL is Extensible

PostgreSQL support several kind of indexes

- BTree, binary tree
- GiST, Generalized Search Tree
- SP-GiST, Space Partitioned GiST
- GIN, Generalized Inverted Index
Binary Tree

Btree, the default index type

- Built for speed
- *unique* concurrency tricks
- Balanced
- support function: `cmp`
- operators: `<= < = > >=`
Generalized Index Search Tree

GiST or the Indexing API

- Built for comfort
- Balanced
- API: consistent, same, union
- API: penalty, picksplit
- API: compress, decompress
- operators: @> <@ &@ = &< &>
  <<| ...
Generalized Inverted iNdex

Indexing several pointers per value, inversed cardinality

- Built for Text Search and Arrays
- Balanced
- API: compare, consistent
- API: extractValue, extractQuery
- operators: @> <@ && =

Dimitri Fontaine

PostgreSQL Extensions

January 31, 2014
Extensions and data types
Some extensions example

46 Contribs, Community extensions, Private ones...

- hll
- cube
- ltree
- citext
- hstore
- earthdistance
- pgq
- pg_trgm
- wildspeed
- plproxy
- PostGIS
- ip4r
- intarray
- prefix
- pgfincore
- pgcrypto
- pg_statstuple
- pg_buffercache
- pg_statstatements
- pgfincore
IP Ranges, ip4r

```
0  7  15  23  31
10010001 00001010 00100010 00000011
```

145 10 34 3

145.10.34.3
### IP Ranges, ip4r

```sql
table geolite.blocks limit 10;
    iprange   | locid
-----------------------+-------
  1.0.0.0/24            | 17
  1.0.1.0-1.0.3.255    | 49
  1.0.4.0/23            | 14409
  1.0.6.0/23            | 17
  1.0.8.0/21            | 49
  1.0.16.0/20           | 14614
  1.0.32.0/19           | 47667
  1.0.64.0/18           | 111
  1.0.128.0-1.0.147.255| 209
  1.0.148.0/24          | 22537
```

(10 rows)
IP Ranges, ip4r, Geolocation

PostgreSQL allows using SQL and JOINs to match IP4R with geolocation.

```
select *
from geolite.blocks
join geolite.location
using(locid)
where iprange
    >>=
'74.125.195.147';
```
IP Ranges, ip4r, Geolocation

PostgreSQL allows using SQL and JOINs to match IP4R with geolocation.

```
select *
from geolite.blocks
join geolite.location
using(locid)
where iprange
    >=
    '74.125.195.147';
```

```
-[ RECORD 1 ]---------------------------------
locid    | 2703
iprange  | 74.125.189.24-74.125.255.255
country  | US
developer | CA
city     | Mountain View
postalcode | 94043
location  | (-122.0574,37.4192)
metrocode | 807
areacode  | 650

Time: 1.335 ms
```
Earth Distance
How Far is The Nearest Pub

The point datatype is in-core

```sql
# CREATE TABLE pubnames
(
    id bigint,
    pos POINT,
    name text
);
```
### How Far is The Nearest Pub

```sql
select name, pos
from pubnames
order by pos <-> point(-6.25,53.346)
limit 3;
```

<table>
<thead>
<tr>
<th>Pub Name</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ned’s</td>
<td>(-6.2519967,53.3458267)</td>
</tr>
<tr>
<td>Sub Lounge</td>
<td>(-6.2542332,53.3469085)</td>
</tr>
<tr>
<td>O’Neill’s of Pearse Street</td>
<td>(-6.2524389,53.3448589)</td>
</tr>
</tbody>
</table>

(3 rows)

Time: 18.679 ms
How Far is The Nearest Pub

```
CREATE INDEX on pubnames USING GIST(pos);

select name, pos 
from pubnames 
order by pos <-> point(-0.12,51) 
limit 3;
```

<table>
<thead>
<tr>
<th>name</th>
<th>pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ned’s</td>
<td>(-6.25,53.34)</td>
</tr>
<tr>
<td>Sub Lo</td>
<td>(-6.25,53.34)</td>
</tr>
<tr>
<td>O’Neil</td>
<td>(-6.25,53.34)</td>
</tr>
</tbody>
</table>

(3 rows)

Time: 0.849 ms
How Far is The Nearest Pub, in Miles please.

```
create extension cube;
create extension earthdistance;

select name, pos <-> point(-6.25,53.34) as miles
from pubnames
order by pos <-> point(-6.25,53.34)
limit 3;
```

<table>
<thead>
<tr>
<th>name</th>
<th>miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ned’s</td>
<td>0.06</td>
</tr>
<tr>
<td>Sub Lo</td>
<td>0.07</td>
</tr>
<tr>
<td>O’Neil</td>
<td>0.12</td>
</tr>
</tbody>
</table>

(3 rows)

Time: 1.335 ms
Some pubs far away from here...

```
select c.name as city, 
    pos @> point(-6.25,53.34) as miles
from pubnames p,
    lateral (select name
        from cities c
        order by c.pos <-> p.pos
        limit 1) c
order by pos <-> point(-6.25,53.34) desc
limit 5;
```

<table>
<thead>
<tr>
<th>city</th>
<th>miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canterbury</td>
<td>399.44</td>
</tr>
<tr>
<td>Canterbury</td>
<td>378.91</td>
</tr>
<tr>
<td>Canterbury</td>
<td>392.08</td>
</tr>
<tr>
<td>Canterbury</td>
<td>397.30</td>
</tr>
<tr>
<td>Canterbury</td>
<td>379.68</td>
</tr>
</tbody>
</table>

(5 rows)

Time: 636.445 ms
Geolocation: ip4r meets earthdistance
with geoloc as

  (  
    select location as l 
    from location 
    join blocks using(locid) 
    where iprange  
      >=  
      '212.58.251.195'
  )

  select name,  
    pos <@> l miles 
  from pubnames, geoloc 
order by pos <-> l 
limit 10;

<table>
<thead>
<tr>
<th>name</th>
<th>miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue Anchor</td>
<td>0.299</td>
</tr>
<tr>
<td>Dukes Head</td>
<td>0.360</td>
</tr>
<tr>
<td>Blue Ball</td>
<td>0.337</td>
</tr>
<tr>
<td>Bell (aka The Rat)</td>
<td>0.481</td>
</tr>
<tr>
<td>on the Green</td>
<td>0.602</td>
</tr>
<tr>
<td>Fox &amp; Hounds</td>
<td>0.549</td>
</tr>
<tr>
<td>Chequers</td>
<td>0.712</td>
</tr>
<tr>
<td>Sportsman</td>
<td>1.377</td>
</tr>
<tr>
<td>Kingswood Arms</td>
<td>1.205</td>
</tr>
<tr>
<td>Tattenham Corner</td>
<td>2.007</td>
</tr>
</tbody>
</table>

(10 rows)

Time: 3.275 ms
Trigrams
Trigrams and similarity

*similar* but not quite *like* the same

```sql
create extension pg_trgm;

select show_trgm('tomy') as tomy,
       show_trgm('Tomy') as "Tomy",
       show_trgm('tom torn') as "tom torn",
       similarity('tomy', 'tom'),
       similarity('dim', 'tom');
```

<table>
<thead>
<tr>
<th>-[ RECORD 1 ]-</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tomy</td>
<td>{&quot;t&quot;,&quot;to&quot;,&quot;my&quot;,&quot;omy&quot;,&quot;tom&quot;}</td>
<td></td>
</tr>
<tr>
<td>Tomy</td>
<td>{&quot;t&quot;,&quot;to&quot;,&quot;my&quot;,&quot;omy&quot;,&quot;tom&quot;}</td>
<td></td>
</tr>
<tr>
<td>tom torn</td>
<td>{&quot;t&quot;,&quot;to&quot;,&quot;om&quot;,&quot;orn&quot;,&quot;rn&quot;,&quot;tom&quot;,&quot;tor&quot;}</td>
<td></td>
</tr>
<tr>
<td>similarity</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>similarity</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Trigrams and typos

Use your data to help your users out

```sql
select actor
  from products
where actor ~* 'tomy';
actor
----------

(0 rows)

Time: <unregistered>
```

```sql
select actor
  from products
where actor % 'tomy';
actor
----------

TOM TORN
TOM DAY

(2 rows)

Time: 26.972 ms
```
Trigrams search indexing

create index on products using gist(actor gist_trgm_ops);

select actor
  from products
  where actor % 'tomy';

actor
----------
TOM TORN
TOM DAY
(2 rows)

Time: 2.695 ms
Trigrams and autocompletion

Use your data to help your users out

```
explain (costs off)
select * from products where actor ~* 'tomy';

QUERY PLAN

Index Scan using products_actor_idx on products
  Index Cond: ((actor)::text ~* 'tomy '::text)
(2 rows)
```
Trigrams and autocompletion

Use your data to help your users out

```sql
select actor
from products
where actor % 'fran'
order by actor <-> 'fran'
limit 10;
```

<table>
<thead>
<tr>
<th>actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRANK HAWKE</td>
</tr>
<tr>
<td>FRANK BERRY</td>
</tr>
<tr>
<td>FRANK POSEY</td>
</tr>
<tr>
<td>FRANK HAWKE</td>
</tr>
<tr>
<td>FRANCES DEE</td>
</tr>
<tr>
<td>FRANK LEIGH</td>
</tr>
<tr>
<td>FRANCES DAY</td>
</tr>
<tr>
<td>FRANK FOSTER</td>
</tr>
<tr>
<td>FRANK HORNE</td>
</tr>
<tr>
<td>FRANK TOMEI</td>
</tr>
</tbody>
</table>

(10 rows)

Time: 2.960 ms
PL/Proxy
PL/Proxy is all about *Sharding*

We’re going to use it for *Remote Procedure Call*
create table example
(
    id serial,
    f1 text,
    f2 text
);

create table audit
(
    change_date timestamptz
      default now(),
    before hstore,
    after  hstore
);

Classic trigger based Auditing

```sql
~# begin;
~*# update example set f1 = 'b' where id = 1;
~*# rollback;

~# select * from audit;
change_date | before | after
-------------+--------+-------
(0 rows)
```
Seting up PL/Proxy

```sql
create extension plproxy;
create server local
  foreign data wrapper plproxy
  options (p0 'dbname=dim');
create user mapping
  for public
  server local
  options (user 'dim');
```
create function test_proxy
    (i int)
    returns int
language plproxy
as 'cluster ''local'';
select i;
';

select test_proxy(1);

test_proxy
-------------
  1
(1 row)

Time: 0.866 ms
Implementing Autonomous Transactions for Auditing

REMOTE TRIGGER
create function audit_trigger()
    returns trigger
    language plpgsql
as ' 
begin
    perform audit_proxy(old, new);
    return new;
end;
';
create function audit_proxy
(
    old example,
    new example
)
returns void
language plproxy
as '.cluster '''local''';
target audit;
';
create function audit
(
    old example,
    new example
)
returns void
language SQL
as ' 
    INSERT INTO audit(before, after)
    SELECT hstore(old), hstore(new);
';
Trigger Definition

```
drop trigger if exists audit on example;

create trigger audit
    after update on example
    for each row
    -- careful, defaults to FOR EACH STATEMENT!
execute procedure audit_trigger();
```
Autonomous Auditing Transaction

```sql
~# begin;
BEGIN

~*# update example set f1 = 'b' where id = 1;
UPDATE 1

~*# rollback;
ROLLBACK
```
Autonomous Auditing Transaction

We did ROLLBACK; the transaction

```
~# select change_date,
    before,
    after,
    after - before as diff
from audit;
- [ RECORD 1 ]--------------------------------
change_date | 2013-10-14 14:29:09.685105+02
before      | "f1"=>"a", "f2"=>"a", "id"=>"1"
after       | "f1"=>"b", "f2"=>"a", "id"=>"1"
diff        | "f1"=>"b"
```
HyperLogLog

State of The Art Cardinality Estimation Algorithm
Creating the unique visitors tracking table

CREATE EXTENSION hll;

-- Create the destination table
CREATE TABLE daily_uniques (
    DATE DATE UNIQUE,
    users hll
);

-- Our first aggregate update
UPDATE daily_uniques
    SET users = hll_add(users,
                       hll_hash_text('123.123.123.123'))
    WHERE date = current_date;
Production ready updates

--
-- First upload a new batch, e.g. using
-- CREATE TEMP TABLE new_batch as VALUES(), (), ...;
--
WITH hll(agg) AS (  
    SELECT hll_add_agg(hll_hash_text(value))  
    FROM new_batch  
)

UPDATE daily_uniques
    SET users = CASE WHEN hll.agg IS NULL THEN users
                    ELSE hll_union(users, hll.agg)  
                    END
    FROM hll
WHERE date = current_date;
with stats as (  
  select date,  
    #users as daily,  
    #hll_union_agg(users) over() as total  
  from daily_uniques  
)  
select date,  
  daily,  
  daily/total*100  
from stats  
order by date;

<table>
<thead>
<tr>
<th>date</th>
<th>daily</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-02-22</td>
<td>401677</td>
<td>25.19</td>
</tr>
<tr>
<td>2013-02-23</td>
<td>660187</td>
<td>41.41</td>
</tr>
<tr>
<td>2013-02-24</td>
<td>869980</td>
<td>54.56</td>
</tr>
<tr>
<td>2013-02-25</td>
<td>154996</td>
<td>9.72</td>
</tr>
</tbody>
</table>

(4 rows)
select to_char(date, 'YYYY/MM'), #hll_union_agg(users) 
from daily_uniques
group by 1;

<table>
<thead>
<tr>
<th>month</th>
<th>monthly</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013/02</td>
<td>1960380</td>
</tr>
</tbody>
</table>

(1 row)
PostgreSQL is YeSQL!
Recap

We saw a number of extensions, each with a practical use case

- **ip4r** IP Ranges and Geolocation
- **Earth** Longitude, Latitude, Computing distances on a map
- **Trigrams** Fixing typos, autocompletion
- **PL/Proxy** Sharding, RPC, Autonomous Transactions
- **HLL** Cardinalities, Unique Visitors
Questions?

Now is the time to ask!