Introduction to PostgreSQL

Ilia Alshanetsky - @iliaa

whois: Ilia Alshanetsky

- PHP Core Developer since 2001
  - Release Master of 4.3, 5.1 and 5.2
- Author of “Guide to PHP Security”
- PostgreSQL user since 2007
- CIO @ Centah Inc.
- Occasional Photographer ;-)

A special thanks to Christophe Pettus whose talk on PostgreSQL was an inspiration for this talk.

Christophe Pettus
PostgreSQL Experts, Inc.
christophe.pettus@pgexperts.com

The Meeting of Elephants

SELECT * FROM PostgreSQL;

- Relational Database Management System
- Open Source answer to Oracle, Microsoft SQL Server, etc…
- Data integrity & reliability is #1 priority
- Free as in Beer (BSD/MIT license)
- It works well, and I like it! ;-)

A Dash of History

- Created by Michael Stonebraker at UC Berkley in 1986
- Developed in secret until 1995
- Becomes an Open Source project in late 1995, as PostgreSQL v6.0
- Rapidly evolving & improving ever since
- Latest stable release is 9.1.3 (use it!)

It runs on?

- All Unixy system types
  - All flavours of Linux, BSDs, OSX, Solaris, etc…
- Windows since v8.0 (2005)
- Does not run on your Blackberry
  - But runs on Android since 2010 ;-)

IN (PostgreSQL)

- More features than you can shake a stick at!
  - You can read the entire, long & boring list at http://www.postgresql.org/about/featurematrix/
- Highly Performant, even on complex queries
- Extremely stable
- Secure (2 critical security bugs in last 5 years)

Auto-Increment via Sequences

PostgreSQL does not have AUTO_INCREMENT, but no worries, it has something even better!

SEQUENCES!

```
CREATE TABLE users (
    id serial,
    name varchar(50),

    primary key(id)
);

postgres=# NOTICE:  CREATE TABLE will create
implicit sequence "users_id_seq" for serial
column "users.id"

postgres=# \d users

                              Table "public.users"
 Column |          Type          |                        Modifiers
--------+------------------------+-----------------------------------------
 id     | integer                | not null default
                                   nextval('users_id_seq'::regclass)
 name   | character varying(50) |
```

```
select setval('users_id_seq', 10);
 setval
---------
      10

select nextval('users_id_seq');
 nextval
---------
      11

select currval('users_id_seq');
 currval
---------
      11
```

```
CREATE SEQUENCE recounter
      INCREMENT BY 2
      MAXVALUE 10
      START WITH 6
      CYCLE;
```

```
SELECT nextval('recounter');          >> 6
SELECT nextval('recounter');          >> 8
SELECT nextval('recounter');          >> 10
SELECT nextval('recounter');          >> 1
```

Data integrity for the WIN

Integrity checks with Constraints

```
CREATE TABLE products (
    id serial,
    name text,
    price numeric CHECK (price > 0),

    primary key(id)
);
```

Go beyond the basic validation

```
CREATE TABLE products (
    id serial,
    name text,
    price numeric CHECK (price > 0),
    discount numeric
      CONSTRAINT discount_check
          CHECK (discount < 0),
          CHECK (-discount < price),

    primary key(id)
);
```

Be the master of your DOMAIN

```
CREATE DOMAIN email_t AS varchar(255)

   CONSTRAINT email_validation

   CHECK (VALUE ~ (
          '^[_a-zA-Z0-9-]+' ||
          E'(\\.[_a-zA-Z0-9-]+)*@[a-zA-Z0-9-]+' ||
          E'(\\.[a-zA-Z0-9-]+)*\\.' ||
          '(([0-9]{1,3})|([a-zA-Z]{2,3})' ||
          '|(aero|coop|info|museum|name))$')
   );

CREATE TABLE users ( … email    email_t …);
```

Oh yes, there are Foreign Keys too

```
CREATE TABLE shopping_cart_items (
    product_id    integer,
    cart_id       integer,
    user_id       integer,

    FOREIGN KEY (product_id)
            REFERENCES products(id),
    FOREIGN KEY (cart_id, user_id)
            REFERENCES shopping_cart(id, user_id)
            ON DELETE CASCADE,
    FOREIGN KEY (user_id)
            REFERENCES users(user_id)
            ON DELETE RESTRICT, ON UPDATE CASCADE
);
```

Not your grandfather's transactions

Transactional Schema Changes

```
BEGIN;

CREATE TABLE foo (id integer);
INSERT INTO foo (id) values(1);

ROLLBACK;

select * from foo;

ERROR:  relation "foo" does not exist
```

Unsure about something? Use Savepoints!

```
BEGIN;

UPDATE wallet SET money = money - 100
                        WHERE user = 'bob';

SAVEPOINT bobs_wallet;

UPDATE wallet SET money = money + 10
                        WHERE user = 'bob';

ROLLBACK TO bobs_wallet;

COMMIT;
```

Multi-Version Concurrency Control

- Uses versioning & snapshots to isolate concurrent queries & update operations.

- Queries see the state of the database at the start of the query or transaction.

- READ COMMITTED (default) or SERIALIZABLE isolation levels

MVCC in Plain English

- Reads don't block writes
- Writes rarely block reads
- Writes don't affect other writes unless both are writing the same data.

And?

- Lot's of concurrent clients are supported easily
- Very little locking happens in most applications
- Transactions are cheap(er)

Data Storage

PostgreSQL provides many different data types for managing your data beyond the basic Numeric, Text and Date types.

Date & Time Interval

```
SELECT '2012-01-12'::date -
  '1 year 2 mons 10 days 5 hours'::interval;
```

```
2010-11-01 19:00:00
```

```
create table foo (bar interval);
insert into foo
      values('11:55:06'::time - '05:22:15'::time);
select * from foo;
```

```
06:32:51
```

Network

```
SELECT '192.168.1.2'::inet;
                        192.168.1.2
SELECT '192.168'::cidr;
                        192.168.0.0/24
SELECT '08002b010203'::macaddr;
                        08:00:2b:01:02:03
SELECT '192.168'::cidr + 300;
                        192.168.1.44/24
```

Arrays

```
CREATE TABLE sample (
    int_list integer[],
    char_box varchar[3][3]
);

INSERT INTO sample (int_list, char_box)
VALUES(
        '{1,2,3,4,5}',
        '{{a,b,c},{x,y,z},{1,2,3}}'
);
```

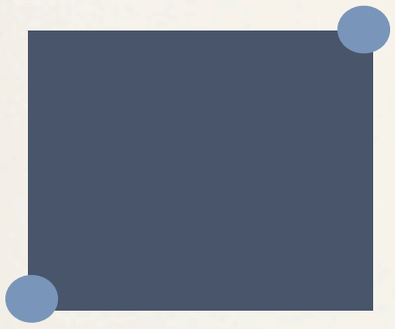
```
  int_list   |          char_box
-------------+-----------------------
{1,2,3,4,5} | {{a,b,c},{x,y,z},{1,2,3}}
```

Geometric

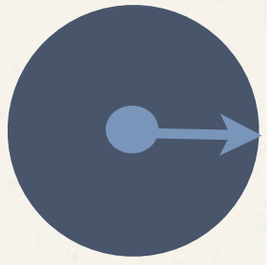
Point (x,y)

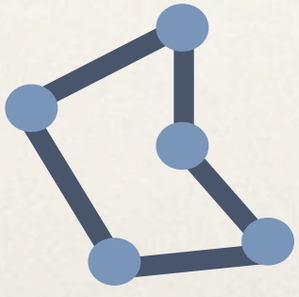
Line ((x1,y1),(x2,y2))

Box ((x1,y1),(x2,y2))



Circle ((x1,y1),r)





Path ((x1,y1),...)

Composite Types

```
CREATE TYPE pair AS (
    x       integer,
    y       numeric
);

CREATE TABLE foo ( id int, bar  pair );

INSERT INTO foo VALUES (1, ROW(1, 5.4));

SELECT * FROM foo;
```

```
 id |   bar
----+---------
  1 | (1,5.4)
```

```
SELECT (bar).x from foo;
```

```
 x
---
 1
```

XML Data Types

```
CREATE TABLE ExMel ( data xml );

INSERT INTO ExMel VALUES

('foo<bar>Baz</bar>'::xml),

(XMLPARSE (CONTENT 'foo<bar>Baz</bar>')),

(XMLPARSE (DOCUMENT '<?xml version="1.0"?><eof/>'));
```

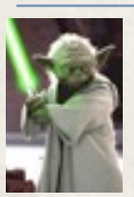
```
                  data
-----------------------
 foo<bar>Baz</bar>
 foo<bar>Baz</bar>
 <eof/>
```

And Many Others...

- UUID
- ENUM
- BIT
- Fulltext (ts_vector & ts_query)
- Boolean
- Bit
- Money
- ByteA

When PostgreSQL install you must, code compile you shall!

200+ config options you will find,



But, panic you will not.

Memory Management

- **shared_buffers** - ideally 25% of total memory

- **work_mem** - 2-10 MB depending on your work-load

- **maintenance_work_mem** - 256 MB for most work-loads

For the rest PostgreSQL will rely on OS disk caching

Risk VS Speed

- **syncronous_commit** - If turned off, don't wait for WAL data to be synced to disk
- **wal_buffers** - 1MB for most installs is plenty
- **checkpoint_segments** - 10 in most cases (every 160MB)
- **fsync** - If turned off, don't sync data to disk (*scary!*)

Other Performance Settings

- **max_connections** - Anything over 300 is case for concern
- **random_page_cost** - 2 on fast disks, < 1 on SSDs
- **default_statistics_target** - 1,000 is a good value in most cases

PHP & PostgreSQL

- PDO::PGSQL
- PgSQL

Making the Connection

```
$db = new PDO(
  "pgsql: dbname=postgres host=localhost" ,
  "postgres" ,
  ""
);

$db = new PDO(
  "pgsql: dbname=postgres host=localhost" .
  " user=postgres password=");
```

Query Execution

```php
$db = new PDO("pgsql: ...");

// create table users
//         (id serial, login text, passwd text);

$db->exec("INSERT INTO users (login, passwd)
                      VALUES('login', 'pwd')");

$db->lastInsertId('users_id_seq');
```

RETURNING Magic

```
$db->query("INSERT INTO users (login, passwd)
                  VALUES('login', 'pwd')
                  RETURNING id"
)->fetchColumn(); // Get record ID
```

```
$db->query("INSERT INTO users (login, passwd)
                  VALUES('login', 'pwd')
                  RETURNING *"
)->fetch(PDO::FETCH_ASSOC); // Get entire record
```

Accumulator

```
$db->query("SELECT array_agg(login) FROM
    (SELECT login FROM users ORDER BY id DESC) q"
)->fetchColumn(); // {bob,mike,jane}
```

```
$db->query("
 SELECT array_to_string(array_agg(login), '<br/>')
    FROM
     (SELECT login FROM users ORDER BY id DESC) q"
)->fetchColumn(); // bob<br/>mike<br/>jane
```

Playing it Safe

Snapshots

```
pg_dump \
    -U postgres \    # login
    -h localhost \   # hostname
    -F c \           # internal format
    -C \             # create database
    -Z 6 \           # compress
    dbname
```

WAL Log Shipping Replication

- Very Simple to Configure
- Allows for a point-in-time data recovery
- On the downside it is not
  - real-time
  - does not offer automatic fail-over

Streaming Replication

- Very simple to setup as of v9.0
- Real-time
- Allows fail-over
- Master-to-slave SELECT query offloading

What about prior releases?

- Slony-I
- PGPool II

Helpful Tools

- pgAdmin
- phpPGAdmin
- psql console (it is awesome!)
- pgFouine

I'd love to hear your feedback

http://joind.in/6001

Ilia Alshanetsky
ilia@ilia.ws
http://ilia.ws/
@iliaa