About
=====
`ngx_postgres` is an upstream module that allows `nginx` to communicate directly
with `PostgreSQL` database.

Response is generated in `rds` format, so it's compatible with `ngx_rds_json`
and `ngx_drizzle` modules.

Status
======
This module is production-ready and it's compatible with following nginx
releases:

- 0.7.x (tested with 0.7.60 to 0.7.69),
- 0.8.x (tested with 0.8.0 to 0.8.55),
- 0.9.x (tested with 0.9.0 to 0.9.7),
- 1.0.x (tested with 1.0.0 to 1.0.11),
- 1.1.x (tested with 1.1.0 to 1.1.12).

Configuration directives
========================
postgres_server
----
* **syntax**: `postgres_server ip[:port] dbname=dbname user=user password=pass`
* **default**: `none`
* **context**: `upstream`

Set details about the database server.

postgres_keepalive
----
* **syntax**: `postgres_keepalive off | max=count [mode=single|multi]
[overflow=ignore|reject]`
* **default**: `max=10 mode=single overflow=ignore`
* **context**: `upstream`

Configure keepalive parameters:

- `max`   - maximum number of keepalive connections (per worker process),
- `mode`   - backend matching mode,
- `overflow` - either `ignore` the fact that keepalive connection pool is full
  and allow request, but close connection afterwards or `reject` request with
  `503 Service Unavailable` response.

postgres_pass
----
* **syntax**: `postgres_pass upstream`
* **default**: `none`
* **context**: `location`, `if location`

Set name of an upstream block that will be used for the database connections
(it can include variables).

postgres_query

```
* **syntax**: `postgres_query [methods] query`
* **default**: `none`
* **context**: `http`, `server`, `location`, `if location`

Set query string (it can include variables). When methods are specified then
query is used only for them, otherwise it's used for all methods.

This directive can be used more than once within same context.


postgres_rewrite
----------------
* **syntax**: `postgres_rewrite [methods] condition [=]status_code`
* **default**: `none`
* **context**: `http`, `server`, `location`, `if location`

Rewrite response `status_code` when given condition is met (first one wins!):

- `no_changes` - no rows were affected by the query,
- `changes`    - at least one row was affected by the query,
- `no_rows`    - no rows were returned in the result-set,
- `rows`       - at least one row was returned in the result-set.

When `status_code` is prefixed with `=` sign then original response body is
send to the client instead of the default error page for given `status_code`.

By design both `no_changes` and `changes` apply only to `INSERT`,
`UPDATE`, `DELETE`, `MOVE`, `FETCH` and `COPY` SQL queries.

This directive can be used more than once within same context.


postgres_output
---------------
* **syntax**: `postgres_output rds|text|value|binary_value|none`
* **default**: `rds`
* **context**: `http`, `server`, `location`, `if location`

Set output format:

- `rds`          - return all values from the result-set in `rds` format
  (with appropriate `Content-Type`),
- `text`         - return all values from the result-set in text format
  (with default `Content-Type`), values are separated by new line,
- `value`        - return single value from the result-set in text format
  (with default `Content-Type`),
- `binary_value` - return single value from the result-set in binary format
  (with default `Content-Type`),
- `none`         - don't return anything, this should be used only when
  extracting values with `postgres_set` for use with other modules (without
  `Content-Type`).


postgres_set
------------
* **syntax**: `postgres_set $variable row column [optional|required]`
* **default**: `none`
* **context**: `http`, `server`, `location`

Get single value from the result-set and keep it in $variable.
```

```
When requirement level is set to `required` and value is either out-of-range,
`NULL` or zero-length, then nginx returns `500 Internal Server Error` response.
Such condition is silently ignored when requirement level is set to `optional`
(default).

Row and column numbers start at 0. Column name can be used instead of column
number.

This directive can be used more than once within same context.


postgres_escape
---------------
* **syntax**: `postgres_escape $escaped [[=]$unescaped]`
* **default**: `none`
* **context**: `http`, `server`, `location`

Escape and quote `$unescaped` string. Result is stored in `$escaped` variable
which can be safely used in SQL queries.

Because nginx cannot tell the difference between empty and non-existing strings,
all empty strings are by default escaped to `NULL` value. This behavior can be
disabled by prefixing `$unescaped` string with `=` sign.


postgres_connect_timeout
------------------------
* **syntax**: `postgres_connect_timeout timeout`
* **default**: `10s`
* **context**: `http`, `server`, `location`

Set timeout for connecting to the database.


postgres_result_timeout
-----------------------
* **syntax**: `postgres_result_timeout timeout`
* **default**: `30s`
* **context**: `http`, `server`, `location`

Set timeout for receiving result from the database.


Configuration variables
=======================
$postgres_columns
-----------------
Number of columns in received result-set.


$postgres_rows
--------------
Number of rows in received result-set.


$postgres_affected
------------------
Number of rows affected by `INSERT`, `UPDATE`, `DELETE`, `MOVE`, `FETCH`
or `COPY` SQL query.
```

$postgres_query
SQL query, as seen by `PostgreSQL` database.

Sample configurations

Sample configuration #1
Return content of table `cats` (in `rds` format).

```
    http {
        upstream database {
            postgres_server  127.0.0.1 dbname=test
                             user=test password=test;
        }

        server {
            location / {
                postgres_pass   database;
                postgres_query  "SELECT * FROM cats";
            }
        }
    }
```

Sample configuration #2
Return only those rows from table `sites` that match `host` filter which
is evaluated for each request based on its `$http_host` variable.

```
    http {
        upstream database {
            postgres_server  127.0.0.1 dbname=test
                             user=test password=test;
        }

        server {
            location / {
                postgres_pass   database;
                postgres_query  SELECT * FROM sites WHERE host='$http_host'";
            }
        }
    }
```

Sample configuration #3
Pass request to the backend selected from the database (traffic router).

```
    http {
        upstream database {
            postgres_server  127.0.0.1 dbname=test
                             user=test password=test;
        }

        server {
            location / {
                eval_subrequest_in_memory  off;

                eval $backend {
```

```
                postgres_pass    database;
                postgres_query   "SELECT * FROM backends LIMIT 1";
                postgres_output  value 0 0;
            }

            proxy_pass  $backend;
        }
    }
}
```

Required modules (other than `ngx_postgres`):

- [nginx-eval-module (agentzh's fork)](http://github.com/agentzh/nginx-eval-module),

Sample configuration #4

Restrict access to local files by authenticating against `PostgreSQL` database.

```
    http {
        upstream database {
            postgres_server  127.0.0.1 dbname=test
                             user=test password=test;
        }

        server {
            location = /auth {
                internal;

                postgres_escape   $user $remote_user;
                postgres_escape   $pass $remote_passwd;

                postgres_pass     database;
                postgres_query    "SELECT login FROM users WHERE login=$user AND
pass=$pass";
                postgres_rewrite  no_rows 403;
                postgres_output   none;
            }

            location / {
                auth_request      /auth;
                root              /files;
            }
        }
    }
```

Required modules (other than `ngx_postgres`):

- [ngx_http_auth_request_module](http://mdounin.ru/hg/ngx_http_auth_request_module/),
- [ngx_coolkit](http://github.com/FRiCKLE/ngx_coolkit).

Sample configuration #5

Simple RESTful webservice returning JSON responses with appropriate HTTP status
codes.

```
    http {
        upstream database {
            postgres_server  127.0.0.1 dbname=test
                             user=test password=test;
        }

        server {
            set $random  123;

            location = /numbers/ {
                postgres_pass     database;
                rds_json          on;

                postgres_query    HEAD GET  "SELECT * FROM numbers";

                postgres_query    POST      "INSERT INTO numbers VALUES('$random')
RETURNING *";
                postgres_rewrite  POST      changes 201;

                postgres_query    DELETE    "DELETE FROM numbers";
                postgres_rewrite  DELETE    no_changes 204;
                postgres_rewrite  DELETE    changes 204;
            }

            location ~ /numbers/(?<num>\d+) {
                postgres_pass     database;
                rds_json          on;

                postgres_query    HEAD GET  "SELECT * FROM numbers WHERE
number='$num'";
                postgres_rewrite  HEAD GET  no_rows 410;

                postgres_query    PUT       "UPDATE numbers SET number='$num' WHERE
number='$num' RETURNING *";
                postgres_rewrite  PUT       no_changes 410;

                postgres_query    DELETE    "DELETE FROM numbers WHERE number='$num'";
                postgres_rewrite  DELETE    no_changes 410;
                postgres_rewrite  DELETE    changes 204;
            }
        }
    }
```

Required modules (other than `ngx_postgres`):

- [ngx_rds_json](http://github.com/agentzh/rds-json-nginx-module).

Sample configuration #6

Use GET parameter in SQL query.

```
    location /quotes {
        set_unescape_uri  $txt $arg_txt;
        postgres_escape   $txt;
        postgres_pass     database;
        postgres_query    "SELECT * FROM quotes WHERE quote=$txt";
    }
```

Required modules (other than `ngx_postgres`):

- [ngx_set_misc](http://github.com/agentzh/set-misc-nginx-module).

Testing

`ngx_postgres` comes with complete test suite based on [Test::Nginx] (http://github.com/agentzh/test-nginx).

You can test core functionality by running:

`$ TEST_NGINX_IGNORE_MISSING_DIRECTIVES=1 prove`

You can also test interoperability with following modules:

- [ngx_coolkit](http://github.com/FRiCKLE/ngx_coolkit),
- [ngx_echo](github.com/agentzh/echo-nginx-module),
- [ngx_form_input](http://github.com/calio/form-input-nginx-module),
- [ngx_set_misc](http://github.com/agentzh/set-misc-nginx-module),
- [ngx_http_auth_request_module](http://mdounin.ru/hg/ngx_http_auth_request_module/),
- [nginx-eval-module (agentzh's fork)](http://github.com/agentzh/nginx-eval-module),
- [ngx_rds_json](http://github.com/agentzh/rds-json-nginx-module).

by running:

`$ prove`

License
=======

This software includes also parts of the code from:

- `nginx` (copyrighted by **Igor Sysoev** under BSD license),
- `ngx_http_upstream_keepalive` module (copyrighted by **Maxim Dounin**
  under BSD license).

See also
========
- [ngx_rds_json](http://github.com/agentzh/rds-json-nginx-module),
- [ngx_drizzle](http://github.com/chaoslawful/drizzle-nginx-module),

- [ngx_lua](http://github.com/chaoslawful/lua-nginx-module),
- [nginx-eval-module (agentzh's fork)](http://github.com/agentzh/nginx-eval-module).