PgREST

PostgreSQL, Javascript, and REST

http://pgre.st/

@clkao PGCON 2014, Ottawa


@clkao

Hi Ottawa, I am from the future!

PgREST is...

- ... a JSON document store
- ... running inside PostgreSQL
- ... working with existing relational data
- ... capable of loading Node.js modules
- ... compatible with MongoLab's REST API
- ... compatible with Firebase API (new!)

PgREST: Existing DB → REST

```
% npm i -g pgrest
% pgrest –db ly
info: Serving `ly` on http://127.0.0.1:3000/
collections
```

```
% curl http://127.0.0.1:3000/collections
["amendments","analytics","bills","calendar",
"ivod","laws","motions","sittings","ttsinterp
ellation","ttsmotions"]
```

```
{
    sitting_introduced: null,
    bill_type: null,
    reconsideration_of: null,
    report_of: null,
    abstract: "行政院函請審議「臺灣地區與大陸地區訂定協議處理及監督條例草案」案。",
    proposed_by: "行政院",
    summary: "函請審議「臺灣地區與大陸地區訂定協議處理及監督條例草案」案。",
    bill_ref: "1374G14948",
    bill_id: "1030403070100200",
  - motions: [
      - {
            sitting_id: "08-05-YS-05",
            resolution: null,
            status: null,
            committee: null,
            motion_class: "announcement",
            agenda_item: 10,
            item: null,
          + dates: […]
        }
    ],
    amendments: [ ],
    cosponsors: null,
    sponsors: null,
  - doc: {
        pdf: "http://lci.ly.gov.tw/LyLCEW/agenda1/02/pdf/08/05/05/LCEWA01_080505_00012.pdf",
        doc: "http://lci.ly.gov.tw/LyLCEW/agenda1/02/word/08/05/05/LCEWA01_080505_00012.doc"
    }
}
```


Relations!

http://api.ly.g0v.tw/v0/collections/bills/1374G14948

NoSQL

Redis

Neo4j
the graph database

CouchDB
relax

Cassandra

mongoDB

membase

riak

Not Only SQL

PostgreSQL 9.2: json type

```
test=# CREATE TABLE x (
    foo json
);
```

Operator	Right Operand Type	Description	Example
->	int	Get JSON array element	`'[1,2,3]'::json->2`
->	text	Get JSON object field	`'{"a":1,"b":2}'::json->'b'`
->>	int	Get JSON array element as text	`'[1,2,3]'::json->>2`
->>	text	Get JSON object field as text	`'{"a":1,"b":2}'::json->>'b'`
#>	array of text	Get JSON object at specified path	`'{"a":[1,2,3],"b":[4,5,6]}'::json#>'{a,2}'`
#>>	array of text	Get JSON object at specified path as text	`'{"a":[1,2,3],"b":[4,5,6]}'::json#>>'{a,2}'`

in postgresql 9.3

PostgreSQL: quick recap

- … Schema (= namespace)
- … View (= predefined queries)
- … Triggers & Rules (= hooking queries)

Schema: namespace

```
test=# CREATE SCHEMA foo;

test=# CREATE TABLE foo.my_table (
    foo json
);

test=# SELECT * from foo.my_table;
```

Can be used for relations, types, functions

Views: Predefined Queries

```
test=# CREATE VIEW foo as
  select * from bar WHERE type = ‘foo’;
```

materialized views in in postgresql 9.3

Triggers/Rules: Hooking Queries

```
CREATE OR REPLACE RULE update_foo AS
  ON UPDATE TO foo
  DO INSTEAD ( … update statement …);
```

BEFORE/AFTER/ON, DO INSTEAD/ALSO

9.1+ has DO INSTEAD support for

plv8

```
CREATE FUNCTION to_jsontext(keys text[], vals text[]) RETURNS text AS
$$
        var o = {};
        for (var i = 0; i < keys.length; i++)
                o[keys[i]] = vals[i];
        return JSON.stringify(o);
$$
LANGUAGE plv8 IMMUTABLE STRICT;

SELECT to_jsontext(ARRAY['age', 'sex'], ARRAY['21', 'female']);
         to_jsontext
------------------------------
 {"age":"21","sex":"female"}
(1 row)
```

Stored Procedures are hard to develop/test/maintain

plv8x

```
test=# select '{"foo": [1,2,3]}'::json |> 'return this.foo[1]';
```

~> ‘@foo[1]’ in coffee

```
test=# CREATE TABLE x (
    foo json
);

test=# SELECT foo ~> ‘this.bar + this.baz’ from x;

test=# UPDATE x SET foo = foo |> ‘delete this.bar; return this’;
```

plv8x with npm modules

```
% npm i qs
% plv8x --import qs
% plv8x -f 'plv8x.json parse_qs(text)=qs:parse'
ok plv8x.json parse_qs(text)

# Now parse_qs is a postgresql function!
test=# select parse_qs('foo=bar&baz=1') as qs;
                  qs
     ------------------------
      {"foo":"bar","baz":"1"}
```

function in PG

pkg:function from npm

functional index

```
test=# CREATE INDEX xfoo on x (_myfunction(mycolumn))
```

```
ly=# \d calendar
Indexes:
    "calendar_pkey" PRIMARY KEY, btree (id)
    "calendar_sitting" btree (_calendar_sitting_id(calendar.*))
```

PgREST

```
% npm i -g pgrest
% pgrest –db ly
info: Serving `ly` on http://127.0.0.1:3000/
collections
```

```
% curl http://127.0.0.1:3000/collections
["amendments","analytics","bills","calendar",
"ivod","laws","motions","sittings","ttsinterp
ellation","ttsmotions"]
```

PgREST

REST Endpoint:

GET, PUT, POST, DELETE

http://localhost:3000/collections/bills


```
{
  - paging: {
        count: 13022,
        l: 30,
        sk: 0
    },
- entries: [
    + {…},
    - {
          sitting_introduced: null,
          bill_type: null,
          reconsideration_of: null,
          report_of: null,
          abstract: "本院台灣團結聯盟黨團，有鑒於近年來臺灣與中國締結協議頻繁，然現行「臺灣地區與大陸地區人
          對於簽署兩岸協議事前、事中與事後之國會監督相關規範多所闕漏，致過去諸多臺灣與中國締結之協議均在簽署之
          查或審議，國會監督流於形式，不符合憲法增修條文 §3 、憲法 §63 及釋#520 所確立之行政院對立法院負責之憲
          權之制衡監督原則及立法院對國家重要事項之參與決策權，實有儘速另於法制化之必要。爰擬具「臺灣與中國締結
          實國會監督，確保立法院參與臺灣和中國締結協議之決策權，並使臺灣與中國締結協議處理規範更加周延。是否有
          proposed_by: "本院台灣團結聯盟黨團",
          summary: "擬具「臺灣與中國締結協議處理條例草案」，請審議案。",
          bill_ref: "1374L16229",
```

PgREST

`http://localhost:3000/collections/bills`

Right,

PgREST is not an ORM!

It’s REST inside PG!

```
ly=# select pgrest_select(‘{
        "collection": “calendar"
      }');
```

PgREST - Queries

`http://localhost:3000/collections/bills`

?q={“summary”:{“$matches”:”foo”}}

Maybe JsQuery in the future

```
ly=# select pgrest_select(‘{
        "collection": “calendar”,
        “q”: {“summary”: {“$matches”: “foo”}}
      }');
```

PgREST - relational

```
CREATE VIEW pgrest.calendar AS
 SELECT _calendar_sitting_id(calendar.*) AS sitting_id,
    calendar.*,
  FROM calendar
  WHERE NOT calendar.ad IS NULL;
```

A declarative way to construct views is also available

PgREST - relational (cont.)

```
'pgrest.calendar':
    f: {-raw}
    s: {date: -1}
    as: 'public.calendar'
    $query: ad: $not: null
    columns:
      sitting_id: $literal:
'_calendar_sitting_id(calendar)'
      '*': {}
```

```
'pgrest.sittings': readonly do
  s: {id: -1}
  f: {-videos}
  as: 'public.sittings'
  primary: \id
  columns:
    '*': {}
    dates:
      $from: 'pgrest.calendar'
      $query: 'sitting_id': $literal: 'sittings.id'
      $order: {date: 1}
      columns:
        'calendar_id': field: 'calendar.id'
        '*': <[chair date time_start time_end]>
  videos:
    $from: 'pgrest.ivod'
    $query: 'sitting_id': $literal: 'sittings.id'
    columns:
      '*': {}
  motions:
    $from: 'pgrest.motions'
    $query: 'sitting_id': $literal: 'sittings.id'
    $order: {motion_class: 1, agenda_item: 1}
```

PgREST - ACL

```
=# CREATE VIEW pgrest.links AS
    SELECT * from links;
```

```
=# CREATE RULE … AS ON UPDATE TO
   pgrest.links DO INSTEAD (
       # ACL check and actual update
   );
```

Ditto for

INSERT / DELETE

PgREST - ACL with token

`http://localhost:3000/collections/bills`

Authorization: Bearer AbCdEf123456

```
ly=# select pgrest_select(‘{
        "collection": “calendar”,
        “pgparam”: {“Authorization”: …}
      }');
```

PgREST - ACL with token

```
=# select pgrest_select('{
        "collection": "calendar",
        "pgparam": {"Authorization": …}
    }');
```

```
=# CREATE RULE … AS ON UPDATE TO
   pgrest.links DO INSTEAD (
      select pgrest_pgparam('Authorization')
      ;UPDATE …
   );
```

PgREST - realtime api

With `pgrest-websocket`

Compatible with firebase api:

```
ref = new PgBase(‘http://localhost:3000/links');

ref.on(“value”, function(snapshot) {…});

ref.set(…);
```

What’s next

- More doc!
- Simplified authn/authz integration
- scaling with londiste/PGQ
- pgbase bindings for frontend frameworks/toolkits
- ngx_postgres <-> postgresql directly
- <Your idea> ?

Thank you!

http://github.com/pgrest/pgrest