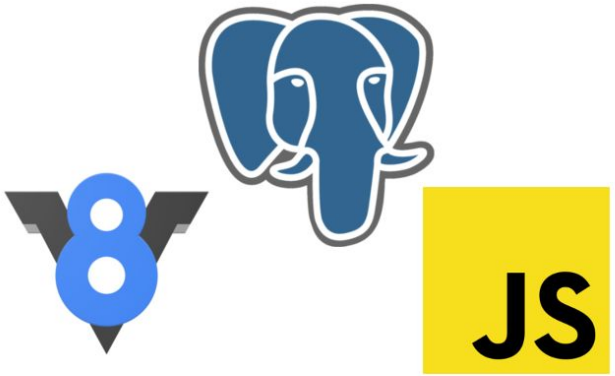
JavaScript inside PostgreSQL



Prague PostgreSQL Developer Day 2020

6 February 2020

About me

Simone Sanfratello

Freelance Software Engineer & Trainer 15y+

JavaScript: node.js, React

PostgreSQL fan since 2001 - v. 7.2

 @simonesanfradev

 Tuscany, Italy

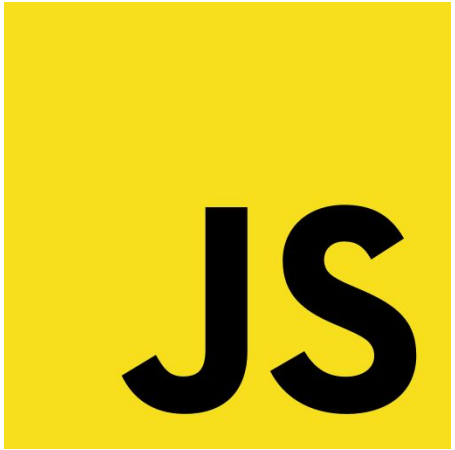
www.braceslab.com

Why ?

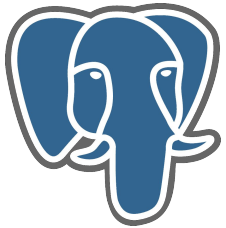
Why using JavaScript?

Developer Experience!

- Functionalities
- Stability
- Documentation
- Tools



What ?

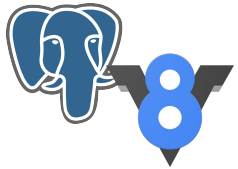


PostgreSQL

- used version: 11.6 (Nov '19) - last version: 12
- https://postgresql.org

v8

- used version: 6.9 (Aug '18) **ES9** - last version: 8.0
- https://v8.dev



plv8

- used version: 2.3.8 (Sep '18) - last version: 2.3.13 | 3.0alpha
- https://plv8.github.io

On AWS RDS since 2018 - pg v. 11 plv8 v. 2.3.8

How ?

Dockerfile example PostgreSQL 11.6 and plv8 2.3.8 extension

```
FROM postgres:11.6 AS plv8
ENV PLV8_VERSION=2.3.8
ENV PLV8_SHASUM="60d7ba9f214ddd4d53c9ec9d004de90bd980aedfec1833cd1ff7286cc4ef796a"
RUN mkdir -p /build
WORKDIR /build
RUN apt-get update && apt-get install -y --no-install-recommends curl ca-certificates p7zip
RUN curl -f -o plv8.7z -SL "https://braceslab.com/public/plv8-${PLV8_VERSION}.7z"
RUN echo ${PLV8_SHASUM} plv8.7z | sha256sum -c
RUN 7zr x plv8.7z

FROM postgres:11.6
ARG PG_USER=pgjs
ARG PG_PASSWORD=pgjs
ARG PG_DB=pgjs
ENV PLV8_VERSION=2.3.8
ENV POSTGRES_USER=${PG_USER}
ENV POSTGRES_PASSWORD=${PG_PASSWORD}
ENV POSTGRES_DB=${PG_DB}
RUN apt-get update && apt-get install libc++1
COPY --from=plv8 /build/plv8-${PLV8_VERSION}.so /usr/lib/postgresql/${PG_MAJOR}/lib/plv8-${PLV8_VERSION}.so
COPY --from=plv8 /build/plv8.control /usr/share/postgresql/${PG_MAJOR}/extension/plv8.control
COPY --from=plv8 /build/plv8--${PLV8_VERSION}.sql /usr/share/postgresql/${PG_MAJOR}/extension/plv8--${PLV8_VERSION}.sql
RUN chmod 0644 /usr/lib/postgresql/${PG_MAJOR}/lib/plv8-${PLV8_VERSION}.so
RUN chmod 0644 /usr/share/postgresql/${PG_MAJOR}/extension/plv8*
```

How ?

enable extension

```
CREATE EXTENSION plv8;
```

here we go!

```
SELECT plv8_version();
```

```
plv8_version
--------------
2.3.8
(1 row)
```

Hands on

Scalar function

```
CREATE FUNCTION to_json(keys TEXT[], values_ TEXT[]) RETURNS JSON AS $$
   const o = {};
   for (let i = 0; i < keys.length; i++) {
     o[keys[i]] = values_[i];
   }
   return o;
$$ LANGUAGE plv8 IMMUTABLE STRICT;
```

```
SELECT to_json(ARRAY['name', 'age'], ARRAY['Tom', '29']);
```

```
to_json
---------------------------
{"name":"Tom","age":"29"}
(1 row)
```

Hands on

Set returning

```
CREATE TYPE my_record AS (i INT, t TEXT);

CREATE FUNCTION set_of_records() RETURNS SETOF my_record AS $$
   plv8.return_next( { "i": 1, "t": "a" } );
   plv8.return_next( { "i": 2, "t": "b" } );
$$ LANGUAGE plv8;

CREATE FUNCTION set_of_records() RETURNS SETOF my_record AS $$
   return [ { "i": 3, "t": "c" }, { "i": 4, "t": "d" } ];
$$ LANGUAGE plv8;
```

```
SELECT * FROM set_of_records();
```

```
i | t
---+---
1 | a
2 | b
3 | c
4 | d
(4 rows)
```

Hands on

Trigger

```
CREATE FUNCTION test_trigger() RETURNS TRIGGER AS $$
    plv8.elog(NOTICE, "NEW = ", JSON.stringify(NEW));
    plv8.elog(NOTICE, "OLD = ", JSON.stringify(OLD));
    plv8.elog(NOTICE, "TG_OP = ", TG_OP);
    plv8.elog(NOTICE, "TG_ARGV = ", TG_ARGV);
    if (TG_OP == "UPDATE") {
        NEW.i = 102;
        return NEW;
    }
$$ LANGUAGE plv8;
```

Hands on

Procedure

```
CREATE TABLE sessions (
    token VARCHAR NOT NULL PRIMARY KEY,
    data JSON,
    expire TIMESTAMPTZ NOT NULL DEFAULT (NOW() + interval '1 month')
);

CREATE PROCEDURE prune_sessions() AS $$
    plv8.elog(INFO, "start pruning sessions");
    const prunes = plv8.execute("DELETE FROM sessions WHERE expire < NOW()");
    plv8.elog(INFO, "pruned", prunes, "sessions");
$$ LANGUAGE plv8;

DO $$
BEGIN
  CALL prune_sessions();
END
$$;
```

JavaScript adaption

- ES9 features: spread operator, rest operator, template string, arrow function and so on
- automapping types between JavaScript and PostgreSQL

  OID, BOOL, INT3, INT4, INT8, FLOAT4, FLOAT8, NUMERIC, DATE,
  TIMESTAMP, TIMESTAMPTZ, BYTEA, JSON, JSONB

- database and log access
- *plv8.find_function*

```
const row_ = plv8.execute(`SELECT * FROM ${table_} WHERE id = $1`, [id])
if (row_.length < 1) {
  plv8.elog(INFO, "no result from table: ", table_, "$id", id)
  return null
}
const [id, ...values_] = row_[0]
return values_
```

Limitations

- can’t import modules (unless compiled within)

- sql statements in *plv8.execute* are just strings > not checked until runtime

- no async, no event loop: it's **procedural**

Case #1: validate data

```
CREATE OR REPLACE FUNCTION validate_email (email TEXT) RETURNS BOOL AS $$
  const exp = /^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+$/
  return exp.test(email)
$$ LANGUAGE plv8 IMMUTABLE STRICT;

CREATE TABLE users (
  id SERIAL NOT NULL PRIMARY KEY,
  ...
  email VARCHAR(128) NOT NULL CHECK (validate_email(email))
);

INSERT INTO users (email) VALUES ('simone@braceslab.com');
INSERT INTO users (email) VALUES ('something');


-- SQL Error [23514]: ERROR: new row for relation "users" violates check
constraint "users_email_check"
-- Detail: Failing row contains (4, something).
```

Case #1: validate data

```
CREATE OR REPLACE FUNCTION test_validate_email() RETURNS void AS $$
  const validate_email = plv8.find_function("validate_email")
  const cases = [
    { email: 'simone@braceslab.com', result: true },
    { email: 'invalid_email', result: false },
  ]
  for (const case_ of cases) {
    const result = validate_email(case_.email)
    if (result != case_.result) {
      throw new Error(`TEST: expected validate_email("${case_.email}") to be
"${case_.result}" but it is ${result}`)
    }
  }
$$ LANGUAGE plv8;
```

Case #1: validate data

```
CREATE FUNCTION validate_sensor_settings (settings  JSON) RETURNS BOOL AS $$
  try {
    if (!settings.name || !settings.measure) {
      return false
    }
    if (settings.checks) {
      const checks = ['daily-limit', 'peek', 'variation']
      if (!Array.isArray(settings.checks) ||
        settings.checks.find(check => !checks.includes(check))) {
        return false
      }
    }
  } catch (error) {
    return false
  }
  return true
$$ LANGUAGE plv8 IMMUTABLE STRICT;
```

Case #2: auditing

```
CREATE TYPE operation AS ENUM ('CREATED', 'UPDATED', 'DELETED');
CREATE FUNCTION create_audit_table (schema_ TEXT, table_ TEXT) RETURNS void AS $$
  plv8.execute(`
  CREATE TABLE ${schema_}.${table_}_audit AS
  SELECT * FROM ${schema_}.${table_}
  WITH NO DATA;
  ALTER TABLE ${schema_}.${table_}_audit
  ADD COLUMN operation operation NOT NULL,
  ADD COLUMN user_id BIGINT NOT NULL,
  ADD COLUMN timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  ADD CONSTRAINT "fk_${schema_}_${table_}_audit_user_id"
    FOREIGN KEY (user_id) REFERENCES auth.users (id)
  `);
$$ LANGUAGE plv8;
```

Case #2: auditing

```
CREATE FUNCTION audit_record (table_ TEXT, id BIGINT, ignore_ TEXT[], operation_
operation, user_id BIGINT) RETURNS void AS $$
  const row = plv8.execute(`SELECT * FROM ${table_} WHERE id = $1`, [id])[0]
  if (!row) {
    plv8.elog(INFO, "no record to audit, table:", table_, "id:", id)
    return
  }
  // ...
  const item = audit_filter(row, ignore_)
  plv8.execute(
    `INSERT INTO ${table_}_audit (operation, user_id,
${audit_columns(item)})
    VALUES ($1, $2, ${audit_markers(item, 3)})`,
    [operation_, user_id, ...audit_values(item)])
  LANGUAGE plv8
```

Case #2: auditing

```
SELECT create_audit_table('bank', 'accounts');

WITH new_account AS (
  INSERT INTO bank.accounts (code) VALUES ('1234') RETURNING id
)
SELECT audit_record ('bank.accounts', new_account.id, array['pin'], 'INSERTED', 1)
FROM new_account;

UPDATE bank.accounts SET amount = 100000 WHERE id = 7;
SELECT audit_record ('bank.accounts', 7, array['pin'], 'UPDATED', 1);

SELECT audit_record ('bank.accounts', 7, array[''], 'DELETED', 1);
DELETE FROM bank.accounts WHERE id = 7;
```

Case #3: formatting

```
CREATE FUNCTION capitalize (str TEXT) RETURNS TEXT AS $$
  return str.toLowerCase()
    // Replaces any - or _ characters with a space
    .replace(/[-_]+/g, ' ')
    // Removes any non alphanumeric characters
    .replace(/[^\w\s]/g, '')
    // Uppercases the first character on each word
    .split(' ')
    .map(s => s[0].toUpperCase() + s.substr(1))
    .join(' ')
$$ LANGUAGE plv8;
```

Case #3: formatting

```
CREATE FUNCTION url_encode (str TEXT) RETURNS TEXT AS $$
  return encodeURIComponent(String(str))
$$ LANGUAGE plv8;

CREATE FUNCTION url_decode (str TEXT) RETURNS TEXT AS $$
  return decodeURIComponent(str.replace(/\+/g, ' '))
$$ LANGUAGE plv8;
```

> https://github.com/google/closure-library/blob/master/closure/goog/string/string.js

Case #4: linked data

```
CREATE FUNCTION user_grants (user_id INT) RETURNS TEXT[] AS $$
  return plv8.execute(
    `SELECT id, code FROM grants
    INNER JOIN users_grants ON grant_id = id AND user_id = $1`,
    [user_id]
  ).map(grant_ => grant_.code)
$$ LANGUAGE plv8;

SELECT users.*, user_grants(users.id)  FROM users;

 id | username |          user_grants
----+----------+-------------------------------
  1 | admin    | {[SYS_ADMIN],[ACCOUNT_EDIT]}
(1 row)
```

Case #5: helper

```
CREATE FUNCTION drop_enum_value (enum_ TEXT, drop_value TEXT, update_value
TEXT) RETURNS void AS $$
  // ... get current values and filter “drop_value”
  plv8.execute(`ALTER TYPE ${enum_} RENAME TO ${enum_}_old`)
  plv8.execute(`CREATE TYPE ${enum_} AS ENUM (${new_values})`)
  // ... get columns that are using the enum
  .forEach(column => plv8.execute(`
    UPDATE ${column.table} SET ${column.name} = '${update_value}'
    WHERE ${column.name} = '${drop_value}';
    ALTER TABLE ${column.table} ALTER COLUMN ${column.name} TYPE ${enum_}
    USING ${column.name}::text::${enum_}`)
  plv8.execute(`DROP TYPE ${enum_}_old`)
$$ LANGUAGE plv8;
```

Case #5: helper

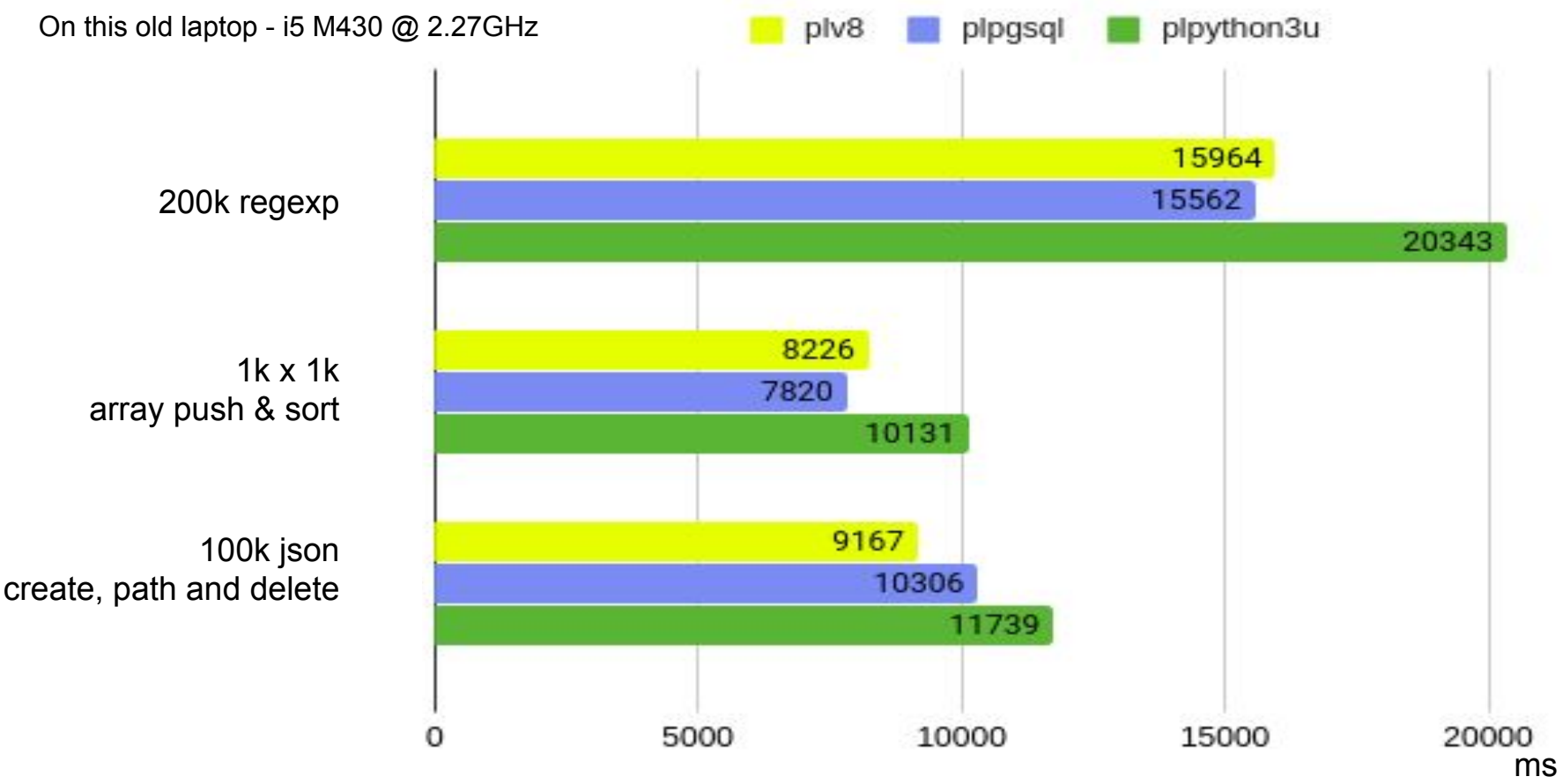
```
CREATE TYPE rate AS ENUM ('A', 'B', 'C', 'D', 'E');

CREATE TABLE ratings (
  id SERIAL NOT NULL PRIMARY KEY,
  evaluation rate NOT NULL,
  notes TEXT
);

SELECT drop_enum_value ('rate', 'E', 'D');
```

Benchmarks

On this old laptop - i5 M430 @ 2.27GHz



Summary

Pros

- rich features and ecosystem
- json native
- dynamic operations
- usual and concise grammar
- performance

Cons

- plv8 is not easy to install (and to compile)
- native PLpgSQL is powerful
- SQL check at runtime

Thank you

Contact me


@simonesanfradev


simone@braceslab.com

www.braceslab.com