Even JSONB In Postgres Needs Schemas!

Controlling Those Unknown Unknowns

POSETTE - 2024

Chris Ellis - @intrbiz@bergamot.social

Hello!

- I’m Chris
  - IT jack of all trades
- Using PostgreSQL ~18 years, across a range of projects:
  - A website search engine
  - UK postal address search, mapping
  - Service Directory
  - Monitoring
  - Smart Energy Analytics and IOT
  - TV, VoD catalogues
  - Booking / Subscriptions

Handling Complex or Unknown Data

Right Tool For The Job?

PostgreSQL JSON Support

```
CREATE TABLE fault_code (
   id              UUID,
   title           TEXT,
   oem_data        JSONB
);
```

PostgreSQL JSON Support

```
INSERT INTO fault_code (oem_data)
VALUES
($${ "plnCode": "PL044"  }$$::JSONB),
($${ "errCode": "E873"   }$$::JSONB),
($${ "crmCode": "CRM114" }$$::JSONB);
```

PostgreSQL JSON Support

```
SELECT *,
   oem_data ->> 'plnCode' AS lineCode,
   oem_data -> 'dims' AS dimensions
FROM fault_code;
```

PostgreSQL JSON Support

```
SELECT *

FROM fault_code

WHERE

   (oem_data ->> 'crmCode') = 'CRM114'

;
```

PostgreSQL JSON Support

```
CREATE INDEX
   fault_code_oem_data_idx
ON fault_code
USING GIN ( oem_data );
```

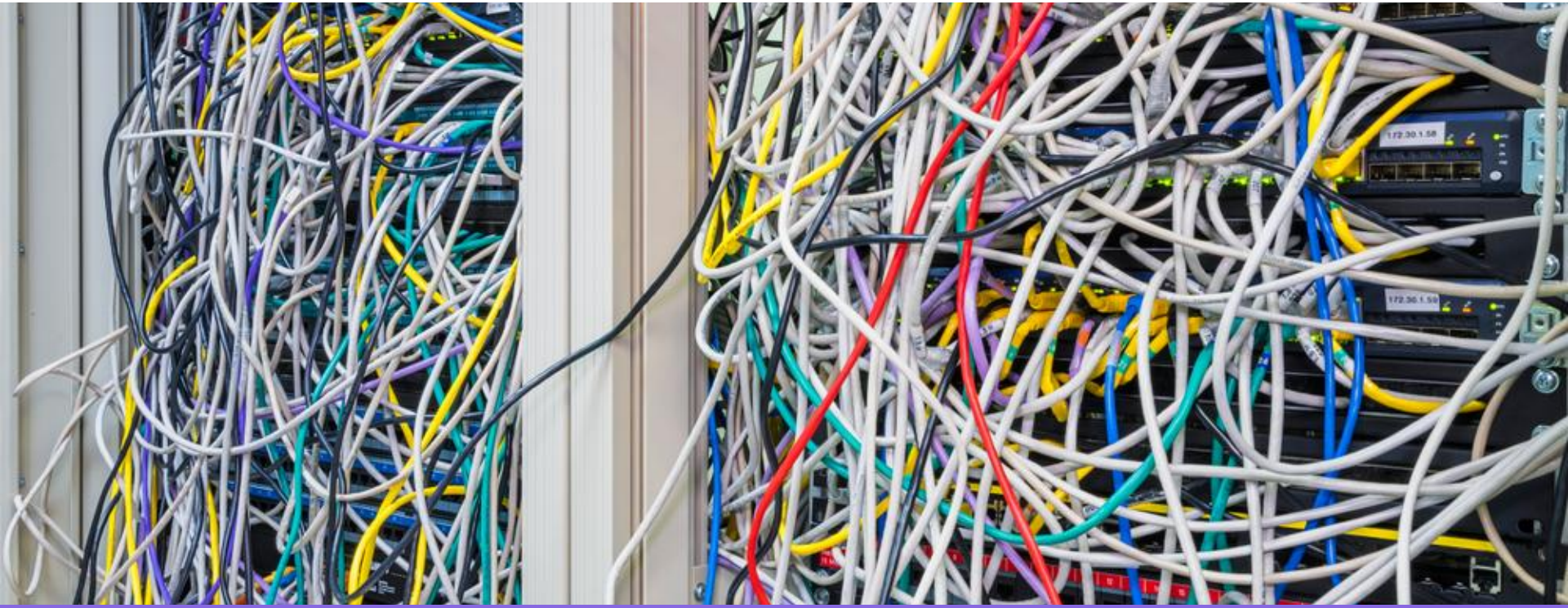
PostgreSQL JSON Support

```
SELECT *

FROM fault_code

WHERE oem_data

    @> $${"crmCode": "CRM114"}$$::JSONB;
```

PostgreSQL JSON Support

```
Bitmap Heap Scan on fault_code
    (cost=28.08..65.25 rows=10 width=99)
    (actual time=0.051..0.054 rows=1 loops=1)
  Recheck Cond: (oem_data @> '{"crmCode": "CRM114"}'::jsonb)
  Heap Blocks: exact=1
  Buffers: shared hit=8
  ->  Bitmap Index Scan on fault_code_oem_data_idx
        (cost=0.00..28.08 rows=10 width=0)
        (actual time=0.033..0.034 rows=1 loops=1)
        Index Cond: (oem_data @> '{"crmCode": "CRM114"}'::jsonb)
        Buffers: shared hit=7
Planning Time: 0.202 ms
Execution Time: 0.140 ms
```

But...



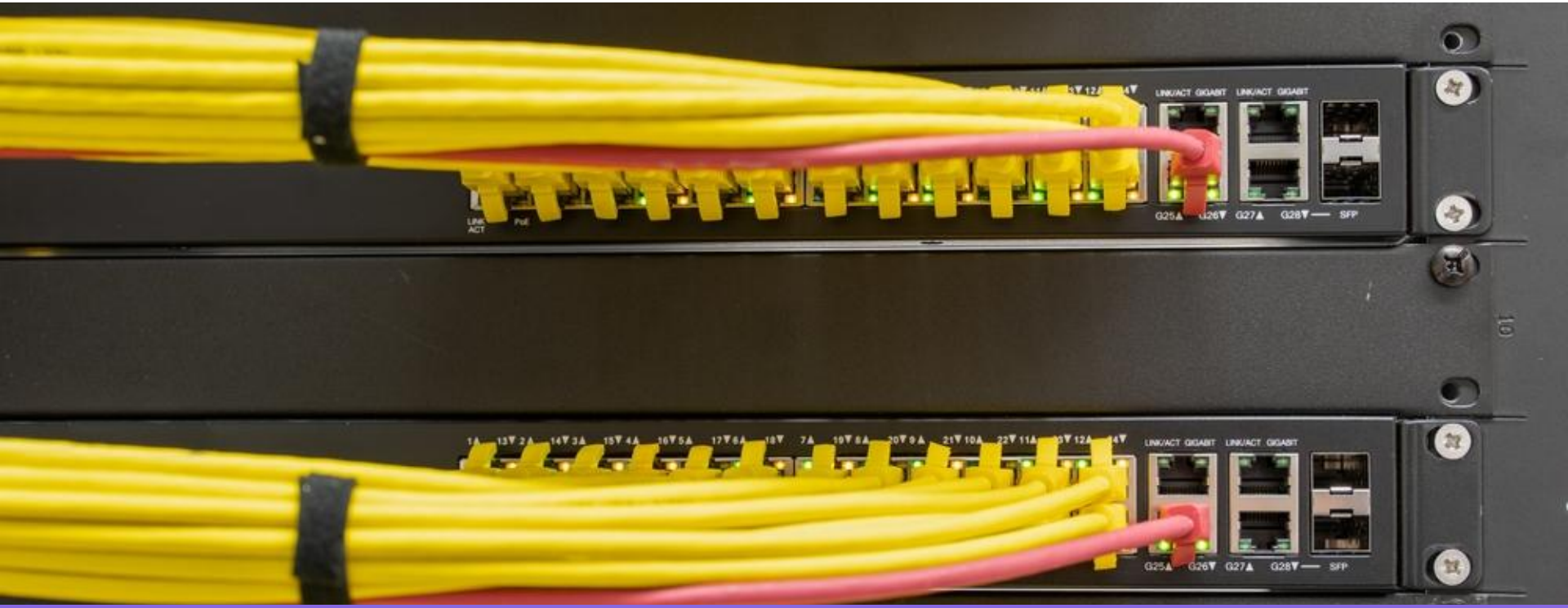
Things Can Go Wrong

```
INSERT INTO fault_code (oem_data)
VALUES
($$[ "a", "b", "c" ]$$::JSONB),
($$0.2523562$$::JSONB),
($$null$$::JSONB);
```

PostgreSQL JSON Support

```
INSERT INTO fault_code (oem_data)
VALUES
($${ "plCode": "PL044"  }$$::JSONB),
($${{ "errCode": "E873"   }}$$::JSONB),
($$[{ "crmCode": "CRM114" }]$$::JSONB);
```

We Want ... Order



Finding The Type

```
SELECT
   jsonb_typeof(
      $$[ "a" ]$$::JSONB
   )
;
```

Quick Cleanup

```
DELETE FROM fault_code

WHERE jsonb_typeof( oem_data )

      <> 'object'

;
```

Check Constraints

```
ALTER TABLE fault_code

ADD CONSTRAINT oem_data_chk

CHECK (

  jsonb_typeof( oem_data ) = 'object'

);
```

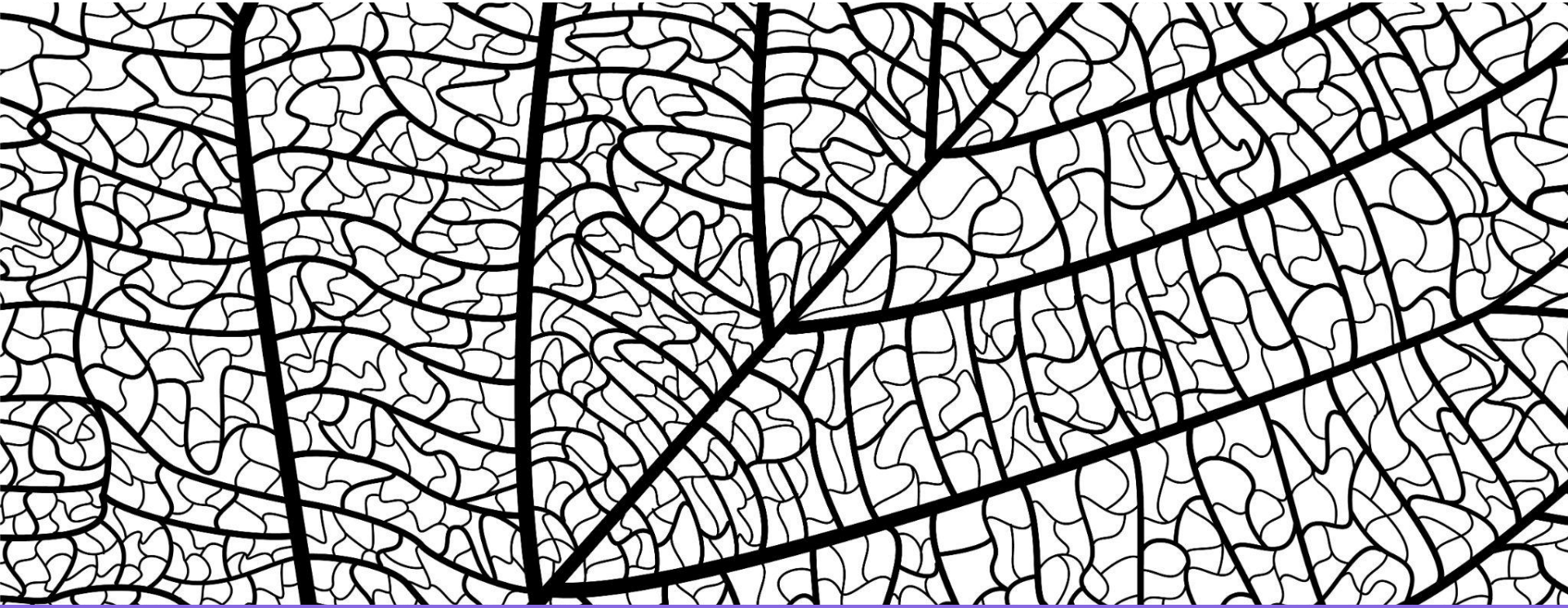
Lets Try Again

```
INSERT INTO fault_code (oem_data)
VALUES
($$[ "a", "b", "c" ]$$::JSONB),
($$0.2523562$$::JSONB),
($$null$$::JSONB);
```

Boom!

```
ERROR:
  new row for relation "fault_code"
  violates check constraint
```

Structure



I Got Some Rulez

```
{
    "type": "rules",
    "decisions": [ {
        "fact": "country",
        "op": "eq",
        "value": "GB"
    } ]
}
```

Checking Structure

```
CHECK (
  jsonb_typeof(ruleset) = 'object' AND
  ruleset -> 'type' IS NOT NULL AND
  jsonb_typeof(ruleset -> 'type') = 'string'
);
```

Going Deeper

```
SELECT
  bool_and( jsonb_typeof(n.e) = 'object' )
FROM jsonb_array_elements(
  ruleset -> 'decisions'
) n(e)
```

Functions To The Rescue

```
CREATE OR REPLACE FUNCTION
    check_ruleset(ruleset JSONB)
RETURNS BOOLEAN LANGUAGE SQL
IMMUTABLE AS $$
  SELECT jsonb_typeof(ruleset) = 'object'
  ...
$$;
```

Functions To The Rescue

```
CHECK (
   check_ruleset( ruleset )
);
```

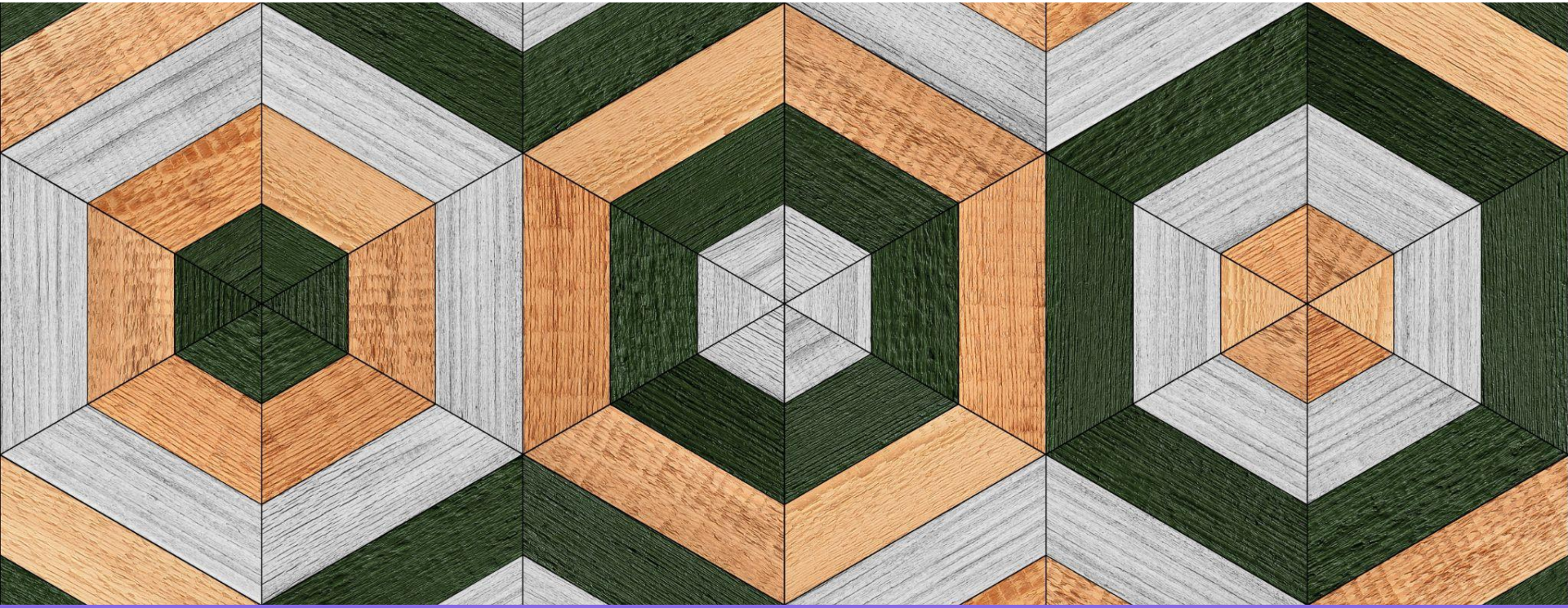
Cooking On Functions

```
CREATE OR REPLACE FUNCTION check_rule(rule JSONB)
RETURNS BOOLEAN LANGUAGE SQL
IMMUTABLE AS $$
  SELECT jsonb_typeof(rule) = 'object'
  AND rule -> 'fact'  IS NOT NULL
  AND rule -> 'op'    IS NOT NULL
  AND rule -> 'value' IS NOT NULL
$$;
```

Cooking On Functions

```
...
AND (SELECT bool_and(check_rule(n.e))
   FROM jsonb_array_elements(
      ruleset -> 'decisions'
   ) n(e)
)
```

So, Back To Those Schemas



We Have Schemas For JSON, Right

```
{
  "type": "object",
  "properties": {
    "type":       {  "type": "string"  },
    "decisions": {  …  }
  },
  "required": ["type", "decisions"],
  "additionalProperties": false
}
```

Enter pg_jsonschema

```
CREATE EXTENSION
    pg_jsonschema;
```

Using pg_jsonschema With Check Constraints

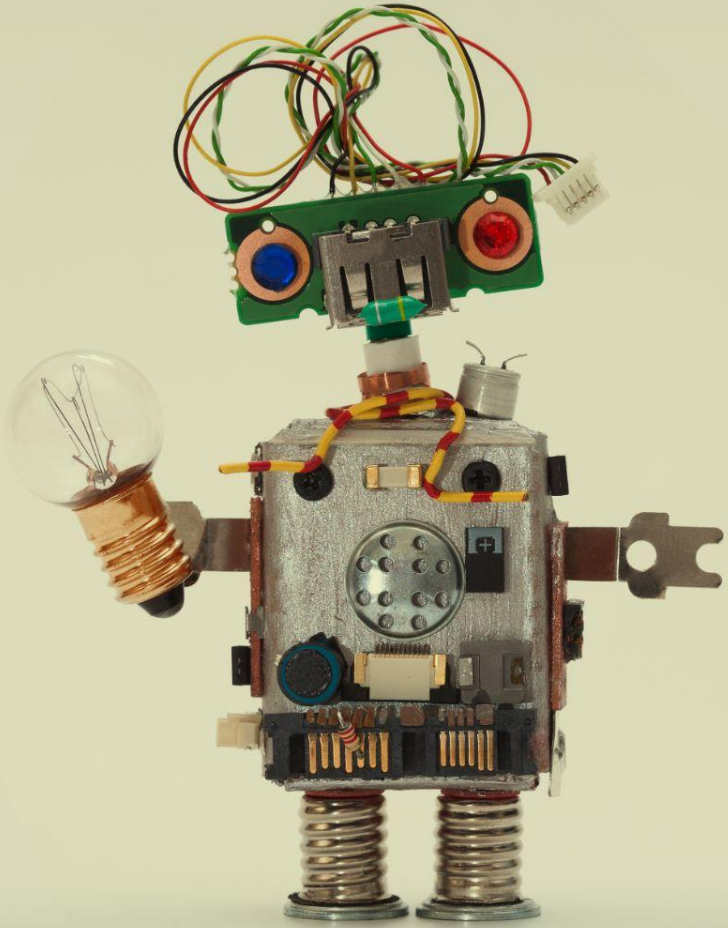
```
CHECK (
  jsonb_matches_schema($${
        "type": "object",
        ...
    }$$::JSON,
    ruleset
  )
);
```

Using pg_jsonschema With Check Constraints

```
CHECK (
  json_matches_schema($${
        "type": "object",
        ...
    }$$::JSON,
    ruleset
  )
);
```

But Extensions ...

https://nexteam.co.uk/pg-jsonschema-gen



Thanks For Listening

Enjoy

POSETTE - 2024