



The power of PostgreSQL exposed with automatically generated API endpoints.

Sylvain Verly

Coderbunker

Development actors



Frontend developer



Backend developer



Database administrator
System administrator

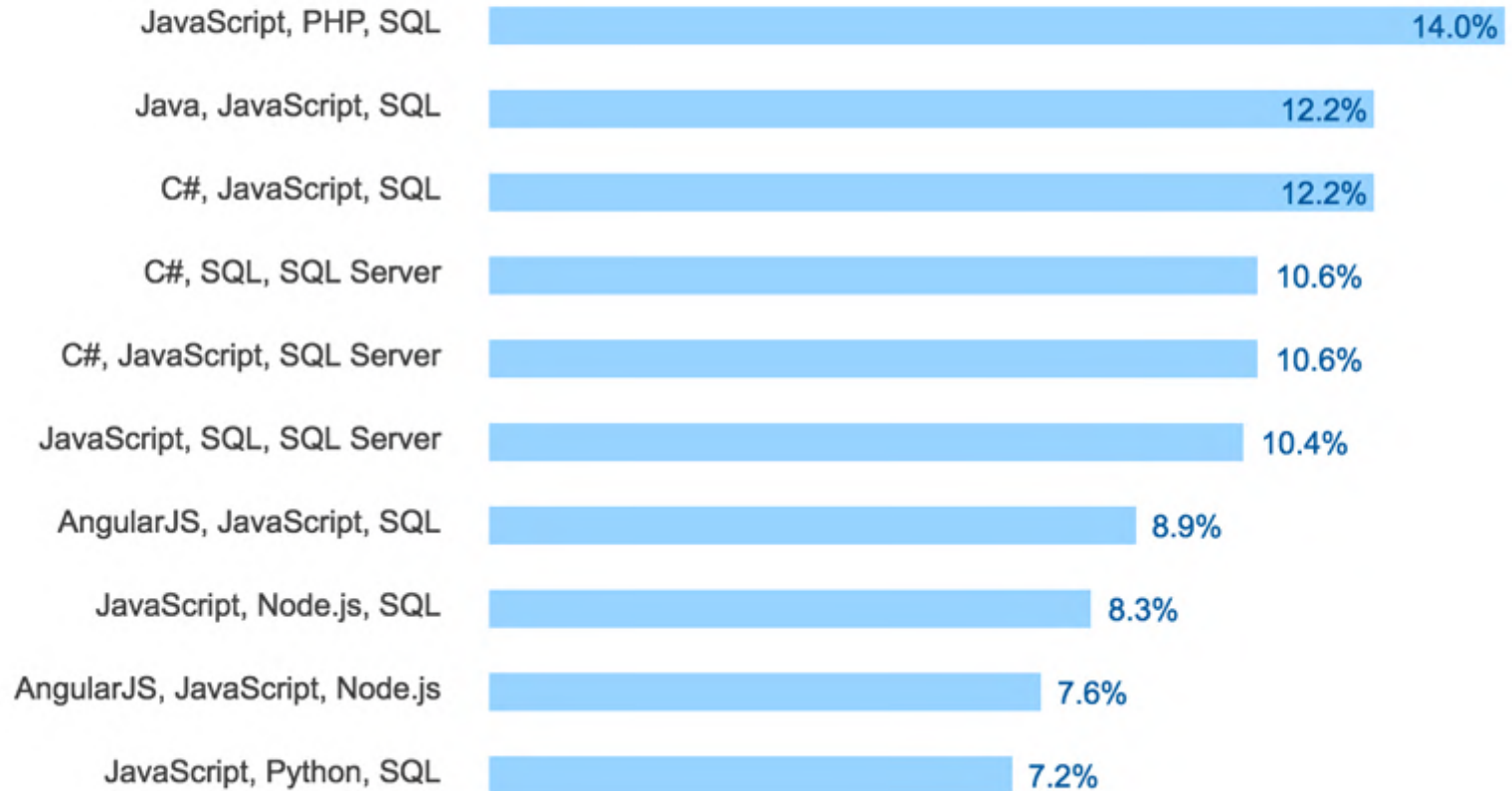


VI. Correlated Technologies

2 Tech

3 Tech

4 Tech



<http://stackoverflow.com/research/developer-survey-2016#technology-correlated-technologies>



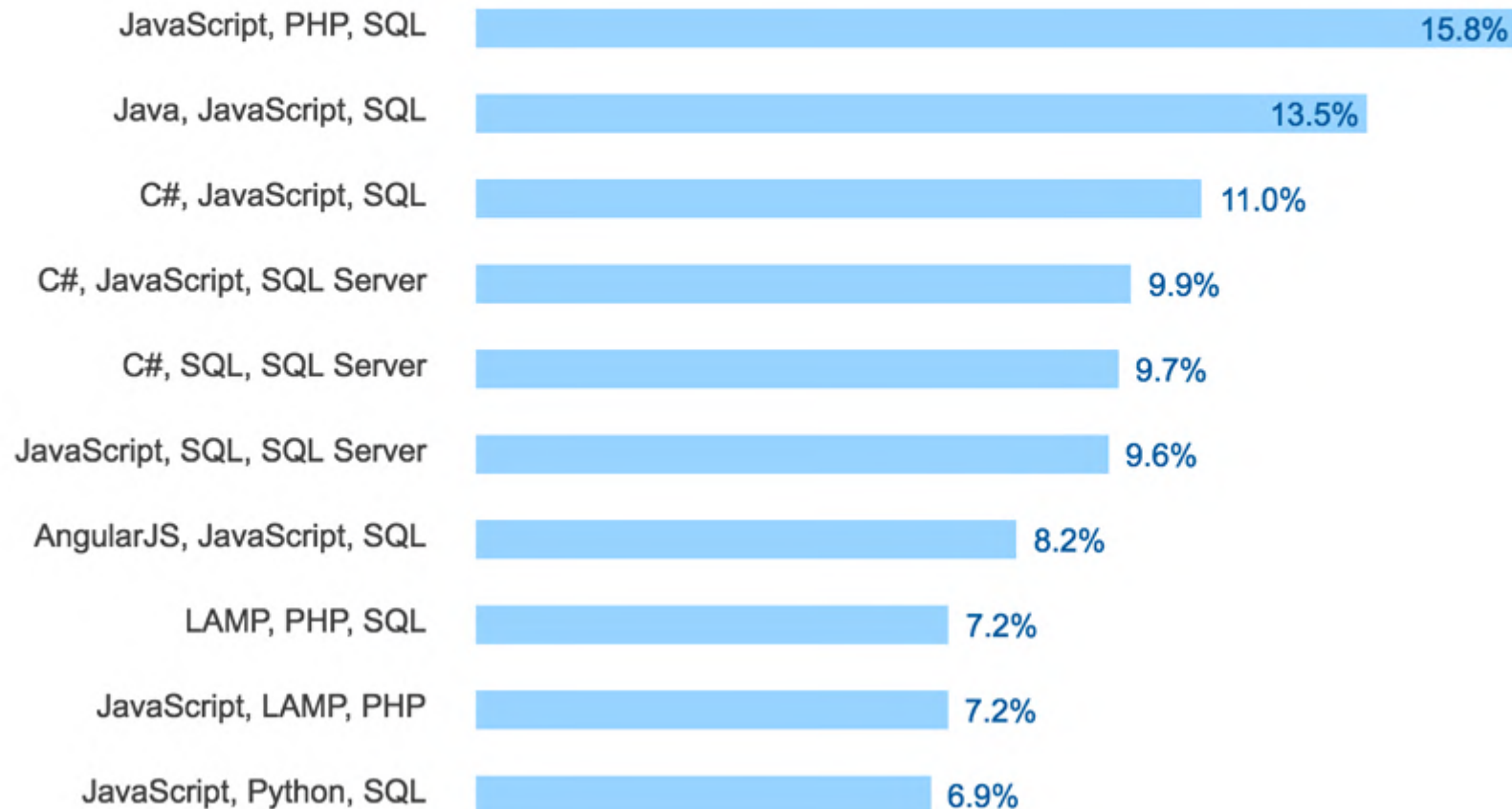
Top Tech Stacks per Occupation

Full-Stack

Front-End

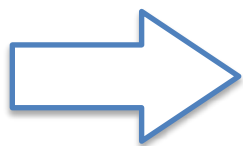
Back-End

Data Scientists



Backend developer

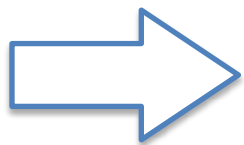
- Don' t like writing SQL
- Interested in speeding up development



- Use Object Relational Mapper
- Describe DB schema using ORM

Database Administrator

- Like writing SQL
- Interested in optimising queries
- Consistent database schema



- Don't use ORM

Database abstraction

- Hides complexity but also key features away,
- Data types are reduced to string, date and numbers,
- Left with a subset of features, a subset of data types.

Database abstraction

- You should know why you are using a specific database: MongoDB, MySQL, etc.
- Not thinking data, schema, data integrity
- Not thinking query optimisation
- Often tied to convention (id primary key)



Example loss of features

- ORM `string(n)` vs `varchar(n)`
- ORM validation vs database constraint



Development

- Do you need ORM ?
 - YES! For library, portable code,
 - NO! You need to know why you are using a database and how to use it.
- Mostly used for CRUD operation
- Development speed increased



Automatic REST API endpoints

- Basic CRUD operations
- No need for mapping
- Bound to your database object
- Forces you to think database:
 - Single point of validation for the data
 - No constrained over ORM convention for database schema

NUODATA

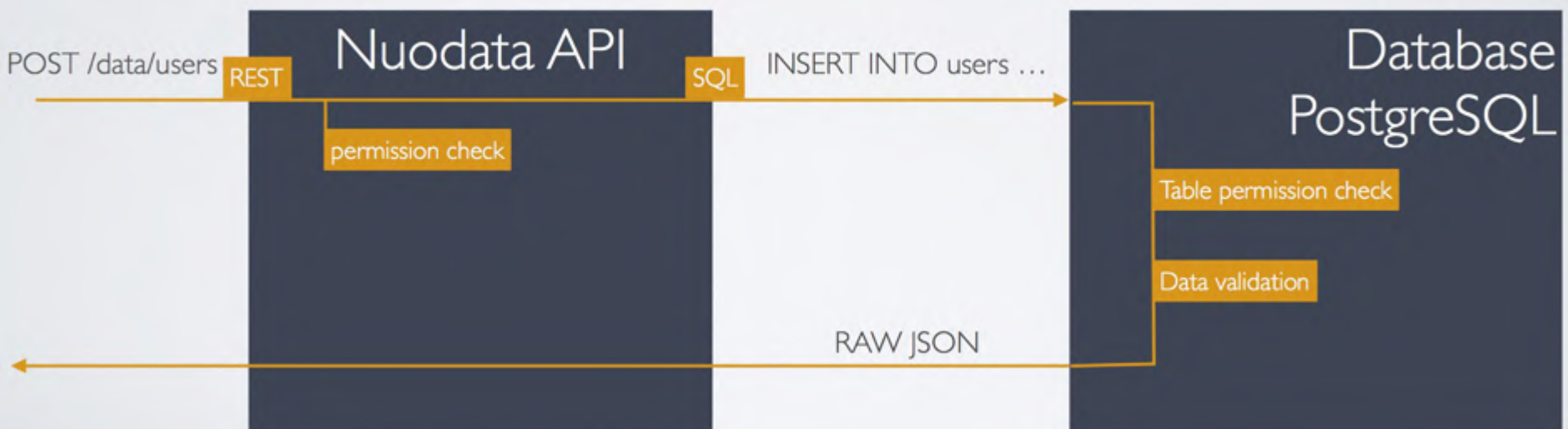
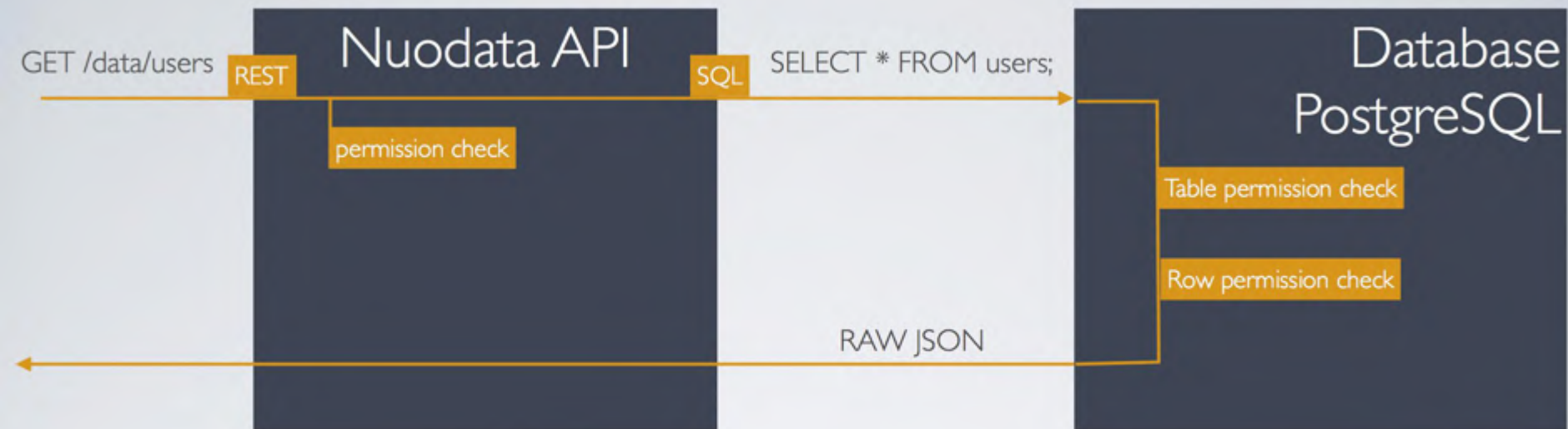
- Automatic REST API endpoint
- Translates REST query into SQL
- Increase speed of development
- Full control of the database:
 - SQL queries
 - DDL statements
 - Architecture Design



Why PostgreSQL ?

- Most flexible database on the market
- PostGIS, JSON, extensions
- Foreign Data Wrapper
- Permissions
- Schema
- Text search
- Notify/Listen





Development speed & quality

- Database exposed via the API
- API is secured with JWT authentication
- You can really focus on your data and its constraints specific to your project.
- Quality data



Internet of Things

- JSON multiple data format
- PostGIS positioning
- Notify / Listen



Internet of Things

```
1 CREATE TABLE thing (  
2   id UUID PRIMARY KEY DEFAULT gen_random_uuid(),  
3   type VARCHAR NOT NULL,  
4   event_data JSONB NOT NULL,  
5   position GEOMETRY NOT NULL,  
6   created_at TIMESTAMP WITH TIME ZONE NOT NULL  
7   DEFAULT CURRENT_TIMESTAMP  
8 );  
9 INSERT INTO thing (type, event_data, position)  
10  VALUES ('temperature', '{"degree":70}', 'POINT(15.2 92.2)');
```

```
HTTP/1.1 POST /data/thing  
Content-Type application/json  
{  
  "type": "temperature",  
  "event_data": {  
    "degree": 70  
  },  
  "position": "POINT(15.2 95.2)",  
  "created_at": "2016-10-16 16:..."  
}
```



Internet of Things

- Notification

```
09 CREATE OR REPLACE FUNCTION notify_event() RETURNS TRIGGER AS $$
10 BEGIN
11     PERFORM pg_notify(NEW.type || '_event', (NEW.event_data)::text);
12     RETURN NEW;
13 END;
14 $$ LANGUAGE 'plpgsql';
15
16 CREATE TRIGGER notify_on_event
17     AFTER INSERT ON thing FOR EACH ROW EXECUTE PROCEDURE notify_event();
18
19 INSERT INTO thing (type, event_data, position)
20     VALUES ('temperature', '{"degree": "35"}', 'POINT(27.2 98.2)');
```

```
HTTP/1.1 GET /subscribe/temperature_event
```

```
Upgrade: websocket
```

```
{
```

```
    "degree" : 70
```

```
}
```



Internet of Things

- Plot on a map

```
24 CREATE OR REPLACE VIEW event_temperature_map
25 AS SELECT st_x(position) as latitude, st_y(position) as longitude, data->>' degree' as temperature
26 FROM thing WHERE type = 'temperature';
```

```
HTTP/1.1 GET /data/event_temperature_map
Content-Type application/json
[ {
  "latitude" : 15.2,
  "longitude" : 95.2
  "temperature" : 70
},
{
  "latitude" : 15.2,
  "longitude" : 95.2
  "temperature" : 20
}]
```



Internet of Things

- Plot statistical data

```
28 CREATE OR REPLACE VIEW event_temperature_count_per_minute
29   SELECT DISTINCT
30     date_trunc('minute', created_at) AS minute,
31     count(*) OVER (ORDER BY date_trunc('minute', created_at)) AS event_per_minute
32   FROM thing;
```

```
HTTP/1.1 GET /data/event_temperature_map
Content-Type application/json
[ {
  "minute" : "2016-10-16 16:44:00+08" : 15.2,
  "count" : 3
},
{
  "minute" : "2016-10-16 16:44:00+08" : 15.2,
  "count" : 2
}, ]
```



Other example use

- Mobile Application
- Single page websites
- Plotting data (e.g. AmCharts js library)
- Plotting geospatial data



Conclusion

- Increased development speed / reduced cost
- Set of basic operations available
- Ease of use with REST API
- Automatic API end point
- Powerfulness of PostgreSQL preserved

Thanks!

Q & A