

# APEX & JSON

Marko Gorički

[marko.goricki@bilog.hr](mailto:marko.goricki@bilog.hr)

[@mgoricki](https://twitter.com/mgoricki)

[apexbyg.blogspot.com](http://apexbyg.blogspot.com)



# BiLog

- small IT company focused on consulting and business solution development (using Oracle technologies)
- big **APEX** company  $\approx$  25 APEX developers
- our products:
  - HRMb - HR management software
  - iRegula - regulatory reporting for insurance



# ...and me?

- I like...

...Oracle + Web technologies = **APEX**

...cycling & running...

...challenges and to push limits...

...and walls :)



# How to...?

There are only five questions about the database:

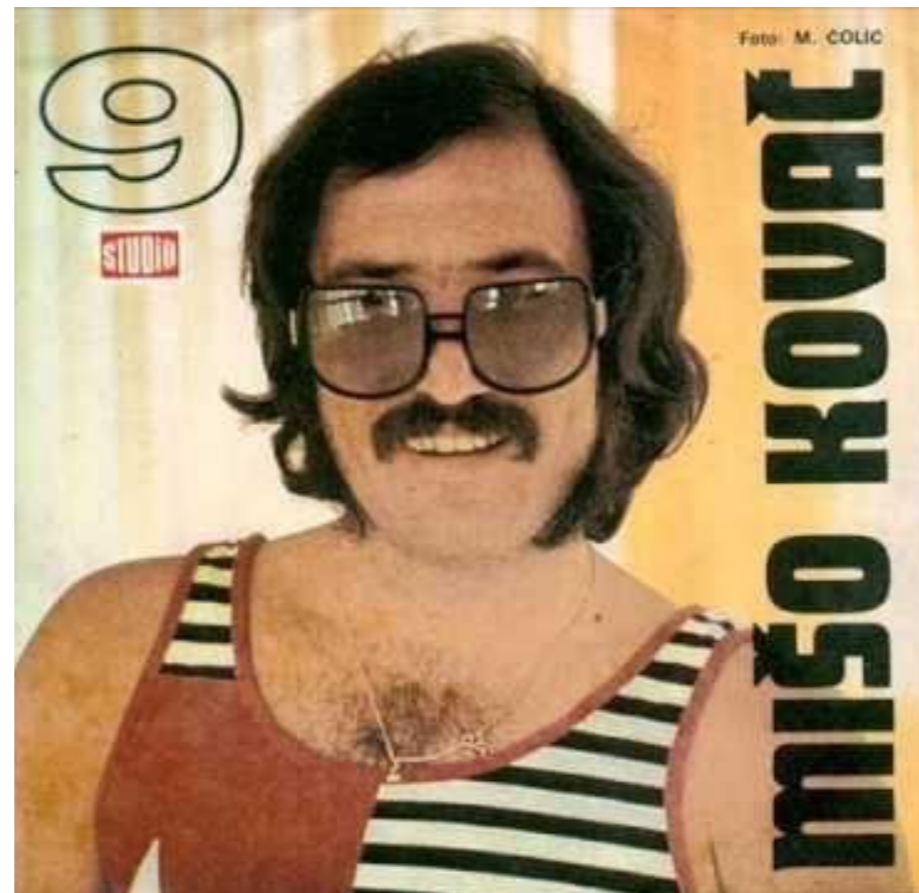
- The answer to the first three questions is “use bind variables.”
- The answer to the fourth question is “do not use bind variables.”
- The answer to the fifth question is “it depends”.

If you know those five answers, you can answer any question about the database!

– Tom Kyte

# Content

- About JSON
- Generate JSON in APEX
- Usage with Mustache.js



# {JSON}

- lightweight data interchange format
- key features:
  - easy to read and write
  - easy to generate and parse
  - language independent

# JSON format

- key elements
  - object
  - array
  - value

```

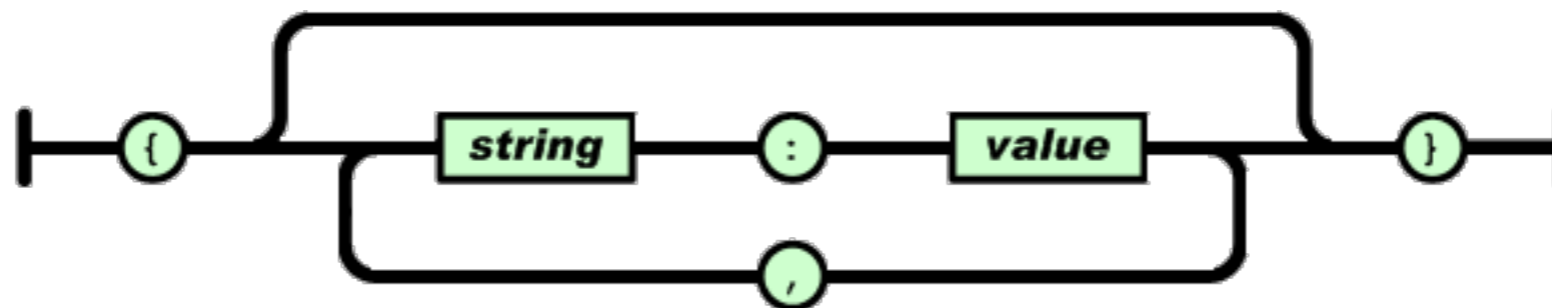
{
  DNAME: "ACCOUNTING",
  LOC: "NEW YORK",
  - EMPLOYEES: [
    - {
      ENAME: "KING",
      JOB: "PRESIDENT",
      HIREDATE: "1981-11-17",
      SAL: 5000,
      hasCommission: false
    },
    - {
      ENAME: "CLARK",
      JOB: "MANAGER",
      HIREDATE: "1981-06-09",
      SAL: 2450,
      MANAGER: "KING",
      hasCommission: false
    },
    - {
      ENAME: "MILLER",
      JOB: "CLERK",
      HIREDATE: "1982-01-23",
      SAL: 1300,
      MANAGER: "CLARK",
      hasCommission: false
    }
  ]
}

```



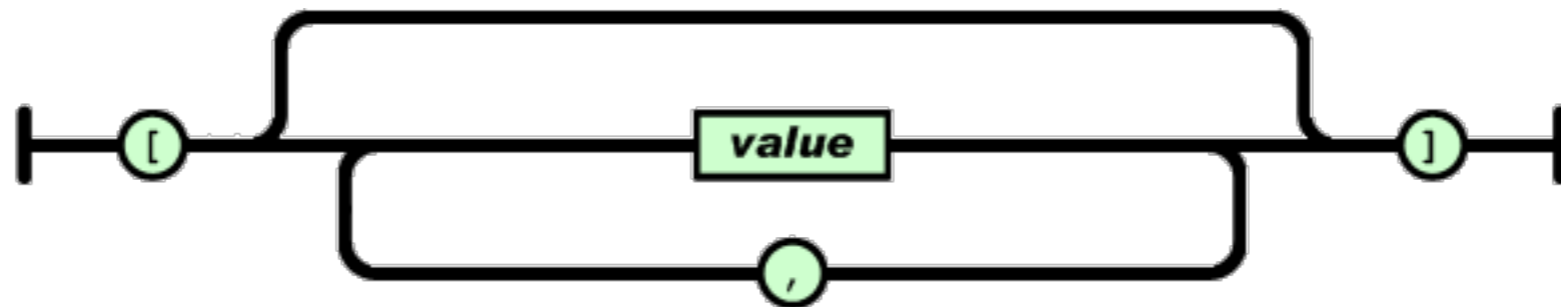
# Object

- unordered set of name/value pairs
- begins and ends with brace - {...}
- name/string followed by colon
- separated by comma



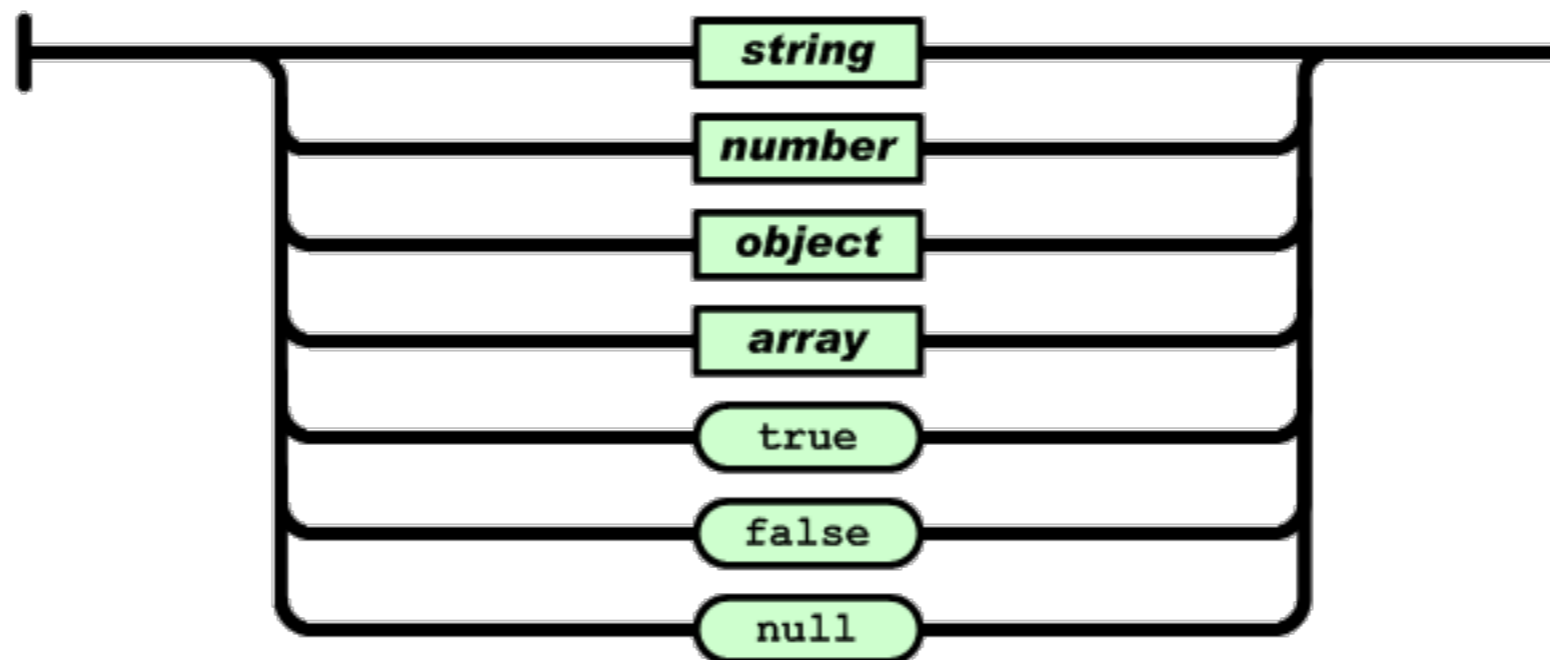
# Array

- ordered collection of values
- begins and ends with brackets - [...]
- separated by comma



# Value

- string - double quotes(“”)
- backslash escapes (\)



# Generate JSON

- prior to APEX 5:
  - by manually concatenating strings
  - apex\_util.json\_from\_\*
  - PL/JSON
- with APEX 5
  - apex\_json package
- Oracle REST Data Services (ORDS)
- node.js

# SQL source

```
select rownum x
      , owner a
      , object_name b
      , object_type c
from all_objects
```

- why short column alias (x, a, b and c)?
  - JSON size:
    - 500 rows - 29.1KB vs 43.3KB
  - less readable vs  $\approx 30\%$  smaller size

# SQL source

Without alias

```

{ ⊖
  "row": [ ⊖
    { ⊖
      "ROWNUM": 1,
      "OWNER": "SYS",
      "OBJECT_NAME": "ORA$BASE",
      "OBJECT_TYPE": "EDITION"
    },
    { ⊖
      "ROWNUM": 2,
      "OWNER": "SYS",
      "OBJECT_NAME": "DUAL",
      "OBJECT_TYPE": "TABLE"
    },
    { ⊕ },
    { ⊕ },
    { ⊕ },
    { ⊕ },
    { ⊕ },
    { ⊕ },
    { ⊕ },
  ],
}

```

With short alias

```

{ ⊖
  "row": [ ⊖
    { ⊖
      "X": 1,
      "A": "SYS",
      "B": "ORA$BASE",
      "C": "EDITION"
    },
    { ⊖
      "X": 2,
      "A": "SYS",
      "B": "DUAL",
      "C": "TABLE"
    },
    { ⊕ },
    { ⊕ },
    { ⊕ },
    { ⊕ },
    { ⊕ },
    { ⊕ },
    { ⊕ },
  ],
}

```

# Manually by concatenating strings

- using sys.htp package
- CONS:
  - hard to maintain and develop
  - manually escape values
  - manually define value types
  - easy to make mistakes
  - no parsing
- PROS
  - customizable
  - it can be fast

```

declare
  v_cnt pls_integer := 0;
  v_tab typ_t_all_objects;
begin
  select rownum x
         , owner a
         , object_name b
         , object_type c
    bulk collect into v_tab
  from all_objects;
  sys.htp.prn('{ "row": [');
  for i in v_tab.first .. v_tab.last
  loop
    if v_cnt = 1 then
      sys.htp.prn(',');
    end if;
    sys.htp.prn('{ "X":' || v_tab(i).x ||
                ', "A":' || v_tab(i).a ||
                ', "B":' || v_tab(i).b ||
                ', "C":' || v_tab(i).c || '"}');
    v_cnt := 1;
  end loop;
  sys.htp.prn('] }');
end;

```

# Using apex\_util.json\_from\*

- procedures: \*\_sql, \*\_array, \*\_items, \*\_string
- CONS:
  - “not documented”
  - no null, true and false values
  - hard to create nested JSON objects
  - no parsing
  - passing string into json\_from\_sql
- PROS:
  - simple and easy to develop
  - auto escaping
  - recognizes some value types (number)
  - available before APEX 5

```

begin
  apex_util.json_from_sql(
    sqlq => 'select rownum x' ||
           ', owner a' ||
           ', object_name b' ||
           ', object_type c' ||
           ' from all_objects');
end;

```



# PL/JSON

- open source library (2009.)
- DB types: JSON (object) and JSON\_LIST (array)
- CONS
  - complex
  - lack of real documentation
  - performance issues
- PROS
  - can be stored in DB
  - parsing
  - supports all value types
  - open source
  - available before APEX 5

```

declare
  obj json;
  obj2 json;
  obj_list json_list;
  v_tab typ_t_all_objects;
begin
  select rownum x
         , owner a
         , object_name b
         , object_type c
  bulk collect
  into v_tab
  from all_objects;
  obj := json();
  obj_list := json_list(); --an empty structure
  obj2 := json();
  for i in v_tab.first..v_tab.last
  loop
    obj_list.append(json({'X': '||v_tab(i).x
                        ||','A': '||v_tab(i).a||''
                        ||','B': '||v_tab(i).b||','
                        ||'C': '||v_tab(i).c||''}
                      ).to_json_value);
  end loop;
  obj.put('row', obj_list);
  obj.htp;
end;

```

# APEX\_JSON API

- PROS:
  - generating and parsing
  - can be used as standalone (CLOB)
  - light footprint
  - native
  - easy conversion to xmltype
- CONS:
  - only in APEX 5.0+
  - generates unnecessary “blanks”

```

declare
  c sys_refcursor;
begin
  open c for
    select rownum x
           , owner a
           , object_name b
           , object_type c
    from all_objects;

  apex_json.open_object;
  apex_json.write('row', c);
  apex_json.close_all;
end;

```

```

...
begin
  apex_json.parse(v_json);
  apex_json.get_varchar2 (p_path => 'row[1].OBJECT_NAME')
end;

```

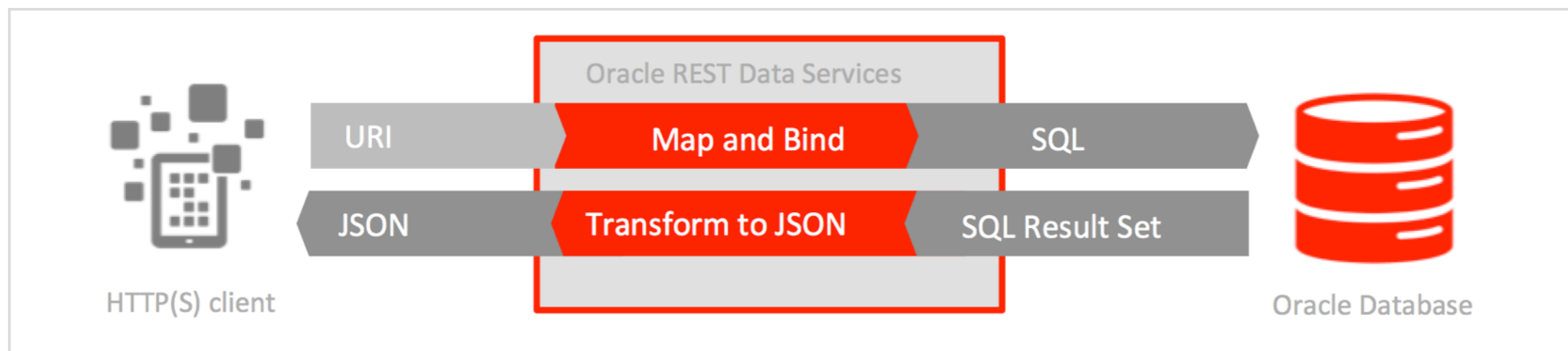
# Oracle REST Data Services (ORDS)

- easy to develop modern REST interfaces
- ways to use it:
  - using APEX GUI
  - SQL Developer (auto-rest)
  - PL/SQL API
  - Simple Oracle Document Access (SODA) API over REST
- URI format:
  - protocol://host:port/ords-name/apex-workspace-name/uri-template/bind-value
  - https://apex.oracle.com/pls/apex/mgoricki/dept/10



select d.  
 , d.lo  
 , curs

from de  
 where d



# Oracle Rest Data Services (ORDS)

Method	GET	⌵	?
Source Type	Query	⌵	?
Format	JSON	⌵	?
Requires Secure Access	No	⌵	?
Pagination Size			?

Source

\* Source ?

```

1 select rownum x, owner a, object_name b, object_type c from all_objects
  
```

[example](#)

URI Template: **dept/{DEPTNO}** ?

 Method  ?

 Source Type  ?

 Requires Secure Access  ?

 Pagination Size  ?

## Source

 \* Source ?

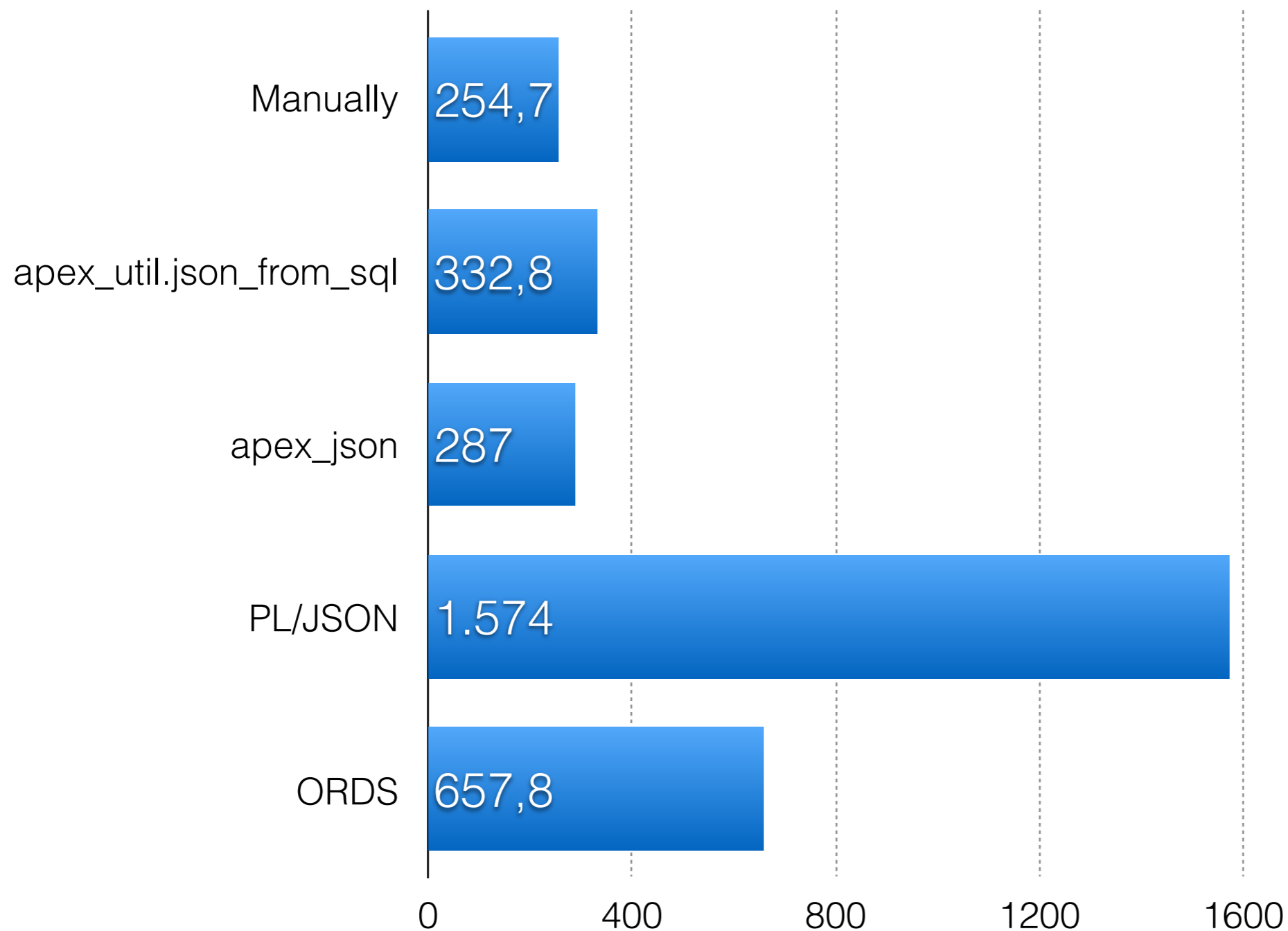
```

1 select d.dname
2     , d.loc
3     , cursor (select e.ename
4                 , e.job
5                 , to_char(e.hiredate, 'yyyy-mm-dd') hiredate
6                 , e.sal
7                 , (select m.ename from emp m where m.empno = e.mgr) manager
8                 , decode(e.comm, null, 'false', 'true') "hasCommision"
9                 from emp e
10                where e.deptno = d.deptno) employees
11 from dept d
12 where d.deptno = :DEPTNO
  
```

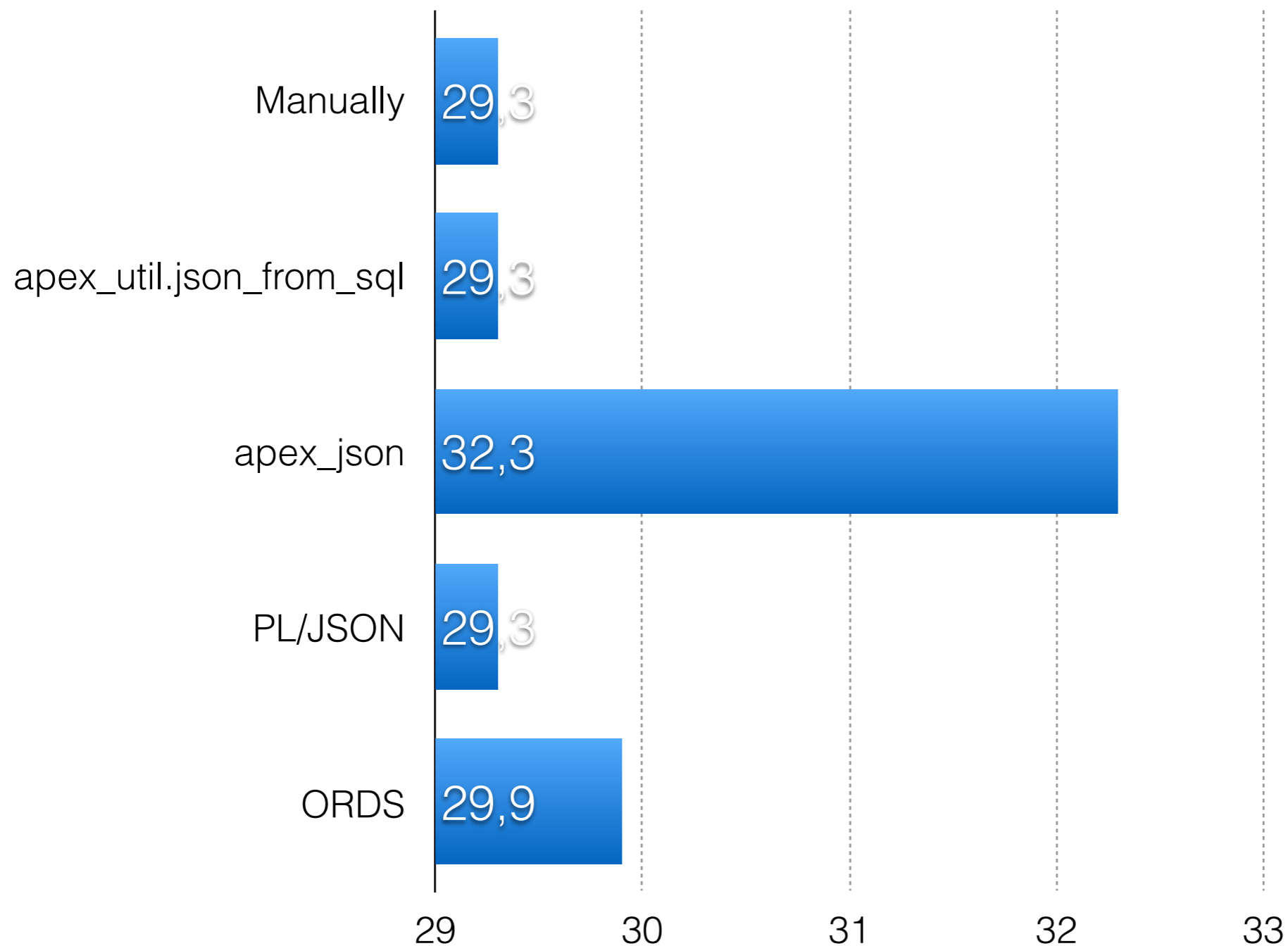
# Oracle Rest Data Services (ORDS)

- CONS
  - little slower
  - you need ORDS
- PROS
  - declarative and simple
  - many options
  - easy to create nested JSON objects

# Average execution time (ms)



# Average JSON size (KB)





# Using JSON

- APEX use it (IR, Page designer)
- fetch asynchronously data with AJAX
- communicate with WS (REST API - Twitter, Flickr, Google Apps)
- easy to use with JavaScript

# Mustache.js

- implementation of Mustache template system in JavaScript
- light weighted, logic-less
- works by expanding tags in a template using values provided by JSON object
- tags can be replaced with value, nothing or series of values



Used by Twitter, LinkedIn, Ebay, PayPal

# Mustache templates

- tags start and end with double mustaches `{{...}}`
- two simple types of tags:
  - variables - basic tag type  
`{{dname}}`
  - sections - render block of text one or more times  
`{{#employees}}...{{/employees}}`

# Mustache templates

```
<ul>
  {{#row}}
  <li>{{X}}, {{A}}, {{B}}, {{C}}</li>
  {{/row}}
</ul>
```

```
<ul>
  <li>1, SYS, ORA$BASE, EDITION</li>
  <li>2, SYS, DUAL, TABLE</li>
</ul>
```

```
{
  "row": [
    {
      "X": 1,
      "A": "SYS",
      "B": "ORA$BASE",
      "C": "EDITION"
    },
    {
      "X": 2,
      "A": "SYS",
      "B": "DUAL",
      "C": "TABLE"
    }
  ]
}
```

```
<ul>
  {{#row}}
  <li>{{X}}, {{A}}, {{B}}, {{C}}</li>
  {{/row}}
</ul>
```

[example](#)

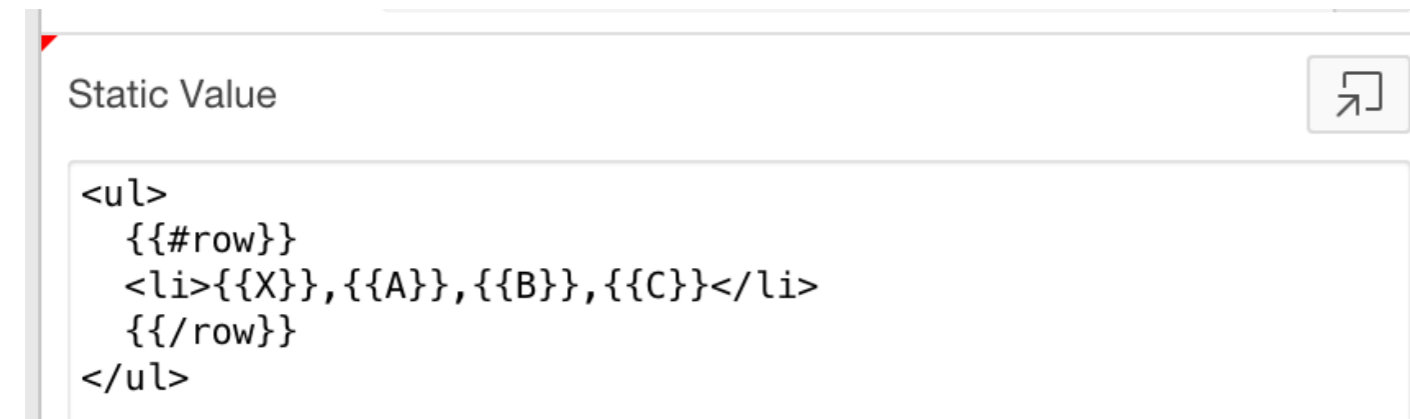
# In APEX

- Add JS library


 A screenshot of the APEX configuration interface for JavaScript. The title is "JavaScript" with a dropdown arrow. Below it is a section labeled "File URLs" with a refresh icon. A text input field contains the value "#APP\_IMAGES#mustache.js".
 

```
JavaScript
File URLs
#APP_IMAGES#mustache.js
```

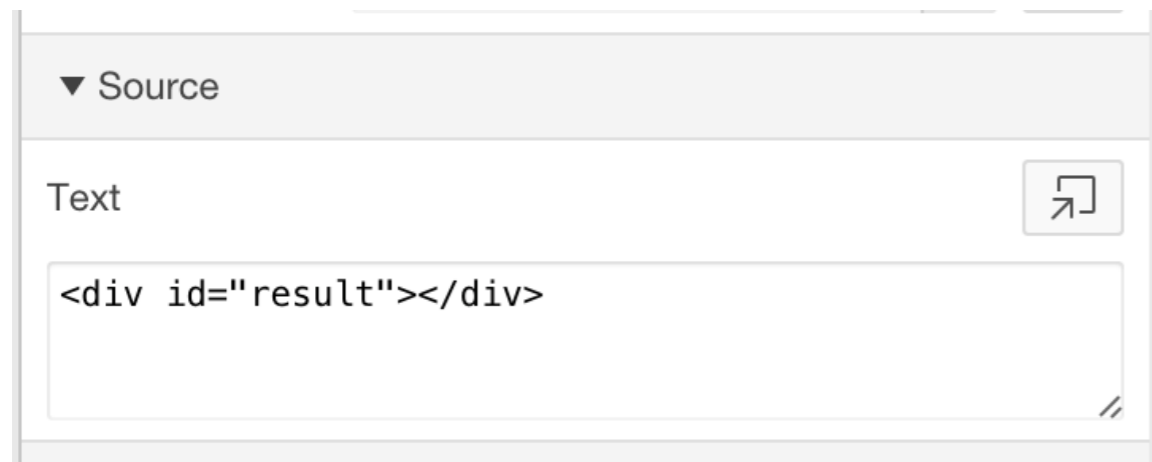
- Create template item


 A screenshot of the APEX configuration interface for a Static Value. The title is "Static Value" with a refresh icon. Below it is a text input field containing HTML code for a list item template.
 

```
Static Value
<ul>
  {{#row}}
  <li>{{X}}, {{A}}, {{B}}, {{C}}</li>
  {{/row}}
</ul>
```

# In APEX

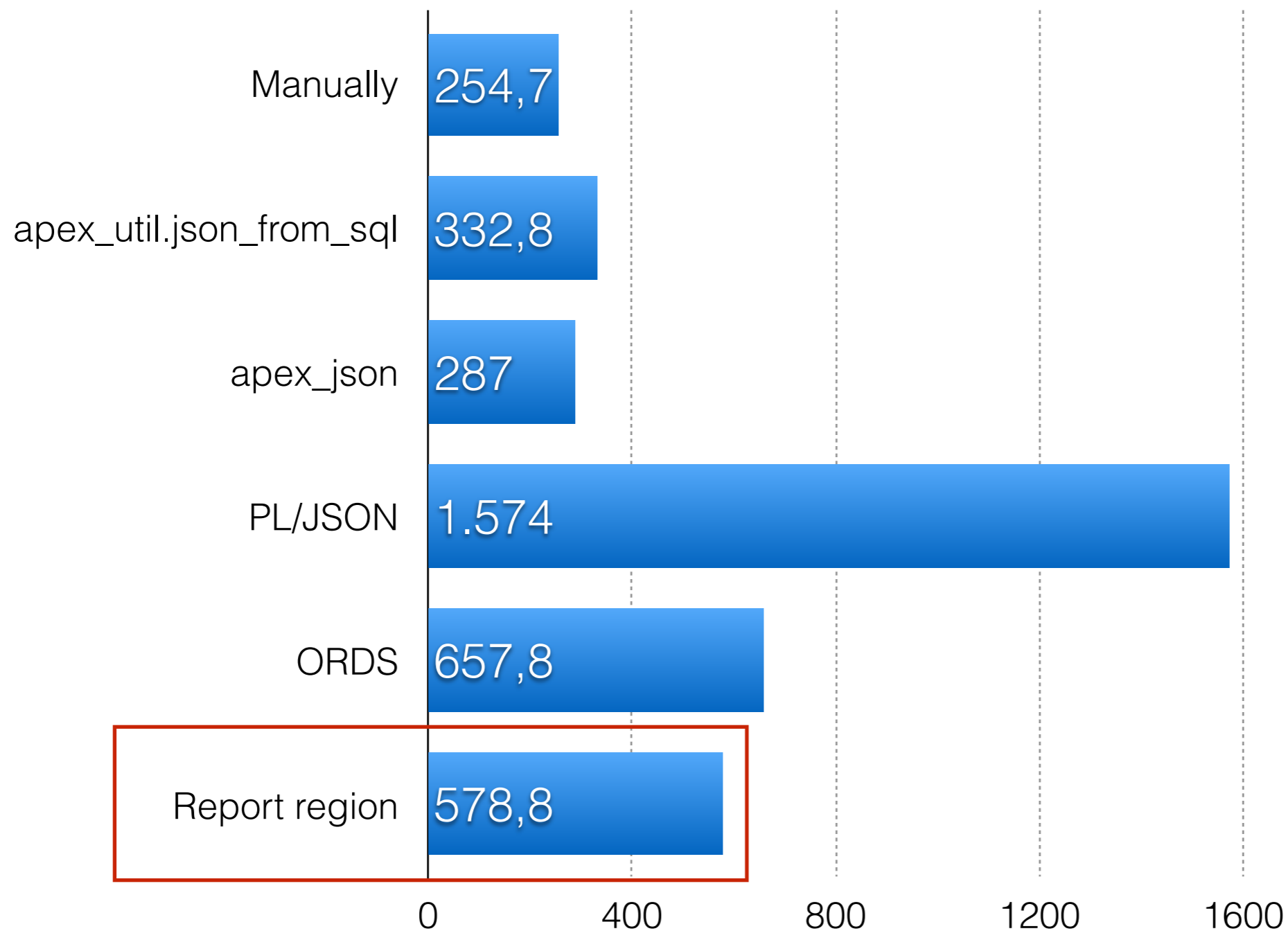
- Define HTML object where to put JSON



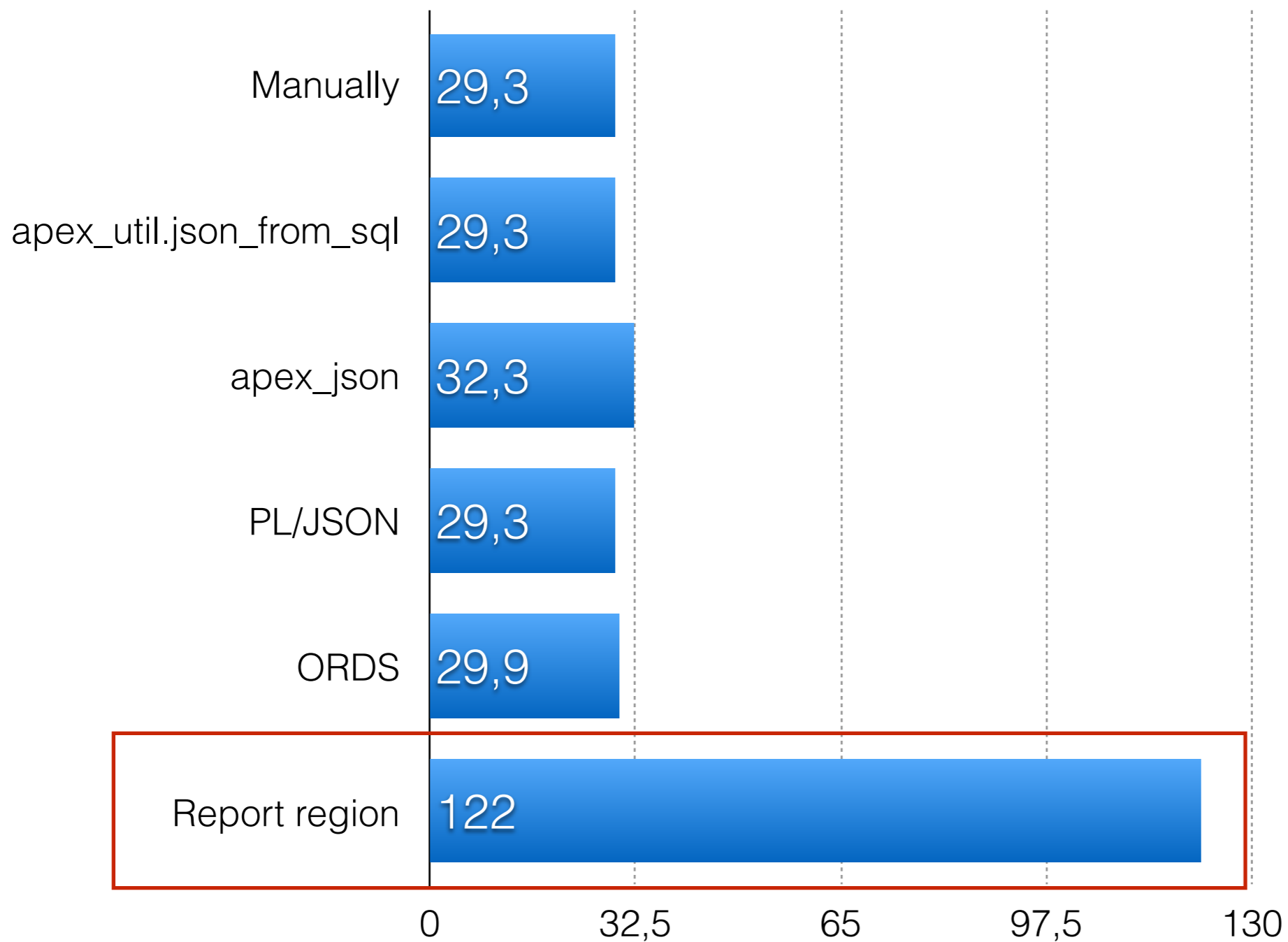
- Fetch JSON and run render function - [example](#)

```
apex.server.process("GET_JSON", {}, {
    success: function(pData) {
        var result = Mustache.render($v('P20_TEMPLATE'), pData);
        $('#result').html(result);
    }
});
```

# Average execution time (ms)



# Average JSON size (KB)





# Q&A

