SQL/JSON

SDRP Lecture 6

Christian Thomsen

[The slides are largely based on the PostgreSQL documention]

Agenda

- JSON basics
- Storing and querying JSON in PostgreSQL
- SQL/JSON path expressions
- From tables to JSON

JSON

- JSON = JavaScript Object Notation
- Open standard for data exchange
- Language independent
- Uses text only, UTF-8 encoded
- Low overhead
  - Shorter/less verbose than XML
    - XML: <SomeTag><AnotherTag>Data</AnotherTag></SomeTag>
  - Easier to parse

- Like XML, JSON
  - is self-describing/human readable
  - supports nesting
  - is widely supported in different programming languages

JSON basics

- A value can be
  - A number – integer or floating point, ***but not*** NaN or Inf
  - A string – in quotes
  - A Boolean – **`true`** or **`false`**
  - **`null`**
  - An ordered array – **`[e1, e2, e3, e4]`**, possibly of different types
  - An object of unordered key/value pairs (where the keys are strings): **`{"k1":v1, "k2":v2, "k3":v3}`**

Example of document:

```
{ "name"    : "Rishi Sunak",
  "address" : { "street" : "Downing Street",
                "number" : 10,
                "city" : "London"
              },
   "age"    : 42
}
```

Storing JSON in your relational database

- Microsoft gives the following reasons:
  - Simplify complex models
  - Store retail and e-commerce data
  - Process log and telemetry ata
  - Store semi-structured IoT data
  - Simplify REST API data
- ISO/IEC argue in TR 19075-6:2017(E)
  - ”technical, business, and government worlds are increasingly using both” [relational and JSON data]
  - “There are great benefits when a single data management system can concurrently handle all of the data”
    - reduced administrative costs
    - improved security and transactions management
    - better performance and greater optimizability
    - better resource allocation
    - increased developer productivity

JSON in PostgreSQL

- PostgreSQL has good support for JSON

- Two types for JSON storage
  - **json**: Stores an exact copy of the input
    (preserves whitespace and order)
  - **jsonb**: Stores in binary format
    (does *not* preserve whitespace and order)
    - Slightly slower to input, much faster to process
    - Indexable

  - We use jsonb in this lecture

Example

```
create table courses(id int, cnt jsonb);
-- using the new jsonb data type

insert into courses values
(1, '{"id":4, "name":"OOP", "semester":2}'),
(2, '{"id":2, "name":"SDRP", "semester":7}');
-- a plain insert statement
-- Note that the id in the JSON is different
-- from the value in the column id!

insert into courses values(3, '{"invalid"');
-- Fails. Good! Would not happen with TEXT
```

Operators

- You can extract an object field or element with ->
- **`select cnt->'name' as name, cnt->'semester' as sem from courses;`**

name	sem
OOP	2
SDRP	7

- For x->n
  - n must be a key name (text) when x is an object
  - n must be an integer when x is an array
- -> gives back jsonb so you can chain ->
  - **`select ('{"a":[9,8,7,6]}'::jsonb)->'a'->0;`** gives the result **9** as jsonb
- If you want text back, you can use ->>
  - **`select ('{"a":[9,8,7,6]}'::jsonb)->'a'->>0;`** gives the result **9** as text

Operators

- With #> you can specify a *path* of keys and array indices to the element to extract
- **`select ('{"a": {"b": ["foo", "bar"]}}' ::jsonb)#>'{a, b, 1}';`** gives ”**bar**” (as jsonb)
- #>> returns text

- **`select '{"a": {"b": ["foo", "bar"]}}'::jsonb ->'no'->'such'->'keys';`**
  and
  `select '{"a": {"b": ["foo", "bar"]}}'::jsonb #>'{no, such, keys}';`
  both return **null** – no error

Existence

- You can check if a string exists as a top-level key or array element with ?
- **`select '{"a":1, "b":2}'::jsonb ? 'b';`**
  gives **true**
- **`select '{"a": ["hello", "world"]}'::jsonb -> 'a' ? 'world';`**
  also gives **true**

- Find the courses where there is info about the TA
  `select cnt from courses where cnt ? 'TA';`

Containment

- You can check if an object is *contained* in another with @>
- The contained object must match the containing object wrt. structure and data contents
  - The containing object may contain more
  - Order of array elements does not matter and duplicates do not matter
- Examples
  - `select '[1, 2, 3]'::jsonb @> '[1, 3]'::jsonb;` is **true**
  - `select '[1, 2, 3]'::jsonb @> '[3, 1]'::jsonb;` is **true**
  - `select '[1, 2, 3]'::jsonb @> '[1, 2, 2]'::jsonb;` is **true**

Containment, cont.

- `select '{"licensePlate": "ABC123", "brand":"Volvo", "owner":{"name": "Bo Xi", "phone":123}}'::jsonb @> '{"brand":"Volvo"}'` is **true**

- `select '{"licensePlate": "ABC123", "brand":"Volvo", "owner":{"name": "Bo Xi", "phone":123}}'::jsonb @> '{"phone": 123}'` is **false**

- `select '{"licensePlate": "ABC123", "brand":"Volvo", "owner":{"name": "Bo Xi", "phone":123}}'::jsonb @> '{"owner":{"phone":123}}'` is **true**

- Often, you will use @> in the WHERE clause
- Find all 7th semester courses:
  - `select cnt from courses where cnt @> '{"semester":7}';`

Querying with subscripting

- Subscripting can be used to extract and modify elements

- `select cnt['name'] as name, cnt['semester'] as sem from courses;`

name	sem
OOP	2
SDRP	7

- `select ('[0,1,2]'::jsonb)[0]` gives **0**
- `select ('[0,1,2]'::jsonb)[-1]` gives **2**
- `select ('[0,1,2]'::jsonb)[3]` gives **[null]**
- `select ('{"a": {"b": {"c": 1}}}'::jsonb)['a']['b']['c']` gives **1**
- `select ('{"a": {"b": {"c": 1}}}'::jsonb)['a']['b']['c']['d']` gives **[null]**

Updating with subscripting

- If **`val['a']`** or **`val['a']['b']`** is not defined, an empty object will be created and filled as necessary
  - The last *existing* element must be an object or array

- **`update courses`**
  `set cnt['TA']['prereqs'] = '"SDRP"'`
  `where id=2`
  will create create an object for “TA” and then add the k/v “prereqs”:“SDRP” to that

Agenda

- JSON basics
- Storing and querying JSON in PostgreSQL
- **SQL/JSON path expressions**
- From tables to JSON

SQL/JSON path expressions

- PostgreSQL also has the type **jsonpath** for efficient binary representation of SQL/JSON path expressions
  - You input the path expressions as SQL strings, i.e., in single quotes
- SQL/JSON path expressions are used to specify items to be retrieved
  - Similar to XPath expressions to retrieve from XML

SQL/JSON path expressions

- A path expression is a sequence of path elements:
  - Literals: Text, numbers, true/false, null
  - Variables:
    - $ is the *context item* and represents the JSON value being queried
    - $*name* is a named variable
    - @ represents a value being filtered
  - Accessors:
    - .*key* gives the object member with that key
    - .* gives the values for all members at the top level of the current obj.
    - [*idx*] gives the single element at that position in an array *(0-based – **not** 1-based as regular SQL arrays, **[last]** also legal)*
    - [*start* **to** *end*] gives an array slice
    - [*] gives all array elements
- Evaluated left to right - an operator deals with the result of the previous step. Parentheses can be added

Example

- `select jsonb_path_query(cnt, '$.name') from courses;`

”OOP”
”SDRP”

- `select jsonb_path_query('{"a":[0,1,2,3]}', '$.a[0 to 1]')`

0
1

Jsonpath operators and methods

- The ”usual” operators: + - * / %
  - Only unary + and – can iterate over multiple values
  - `select jsonb_path_query('{"a":[0,1,2,3]}', '$.a[0] + 1')` works
  - `select jsonb_path_query('{"a":[0,1,2,3]}', '$.a + 1')` **fails**
- .size() → size of array or 1 if not on an array
- .type() → ”string”, ”object”, …
- .double()
- .ceiling()
- .floor()
- .abs()
- .datetime() – converts a string to date/time value
  - Order of attempts: date, timetz, time, timestapmptz, timestamp

Filters

- Filter expressions can be used – similar to SQL WHERE
- ? (condition)

- Written immediately after the step to apply the filter to
  - @ denotes the value being filtered coming from the prev. step
- **`select jsonb_path_query(cnt, '$ ? (@.semester >= 6).name') from courses;`**
  gives **“SDRP”**

- You can of course have more filters:
  - $? (@.x < 10) ? (@.y < 20)
  - $? (@.x < 10).y ? (@ < 20)
  - $? (@.x < 10 && @.y < 20)

Filters, cont.

- The expected operators can be used: ==, !=, <>, <, >, <=, >=, &&, ||, !

In addition

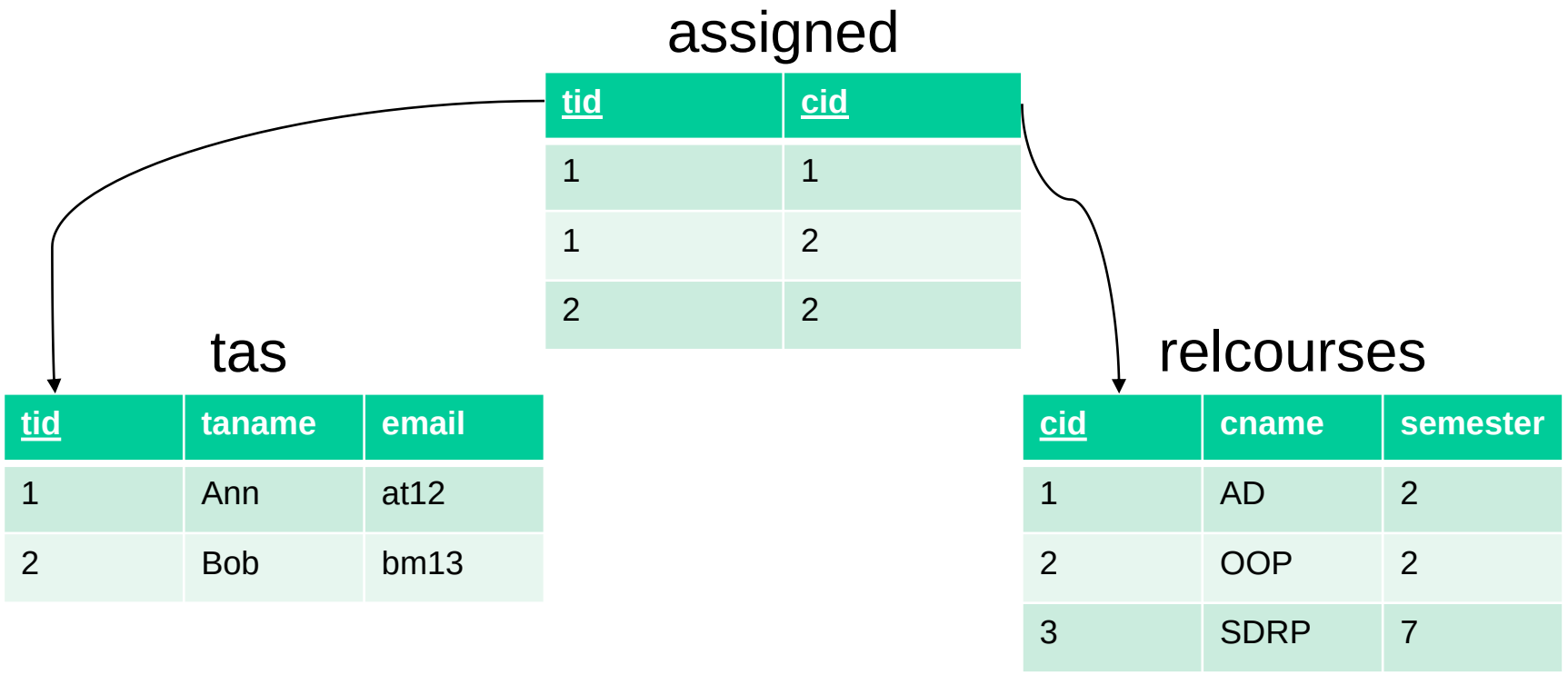
- **is unknown**:
  `jsonb_path_query('[-1, 2, 7, "foo"]', '$[*] ? ((@ > 0) is unknown)')` gives **"foo"**
- **starts with**:
  `jsonb_path_query('["John Smith", "Ann Ai"]', '$[*] ? (@ starts with "John")')`
  gives **"John Smith"**
- **exists**:
  `select jsonb_path_query(cnt, '$ ? (exists (@.TA)).name') from courses;` gives **"SDRP"**

Agenda

- JSON basics
- Storing and querying JSON in PostgreSQL
- SQL/JSON path expressions
- **From tables to JSON**

From tables to JSON

- Much data is stored in relational databases, but you may need to exchange that with someone else
- You can create JSON from your relational data
- Setting the stage for our example:



assigned

tid	cid
1	1
1	2
2	2

tas

tid	taname	email
1	Ann	at12
2	Bob	bm13

relcourses

cid	cname	semester
1	AD	2
2	OOP	2
3	SDRP	7

Simple start

- `select to_jsonb(relcourses.*) from relcourses;`

to_jsonb
"{""cid"": 1, ""cname"": ""AD"", ""semester"": 2}"
"{""cid"": 2, ""cname"": ""OOP"", ""semester"": 2}"
"{""cid"": 3, ""cname"": ""SDRP"", ""semester"": 7}"

- So far so good, but we we lack information about the TAs
- `select jsonb_build_object('name', taname, 'email', email) from tas;`

jsonb_build_object
"{""name"": ""Ann"", ""email"": ""at12""}"
"{""name"": ""Bob"", ""email"": ""bm13""}"

Courses and TAs

- ```
  with
  tainfo as (select tid, jsonb_build_object('name',
              taname, 'email', email) as ta from tas),
  coursesandtas as (select cname, semester, ta
                from relcourses natural left outer join
                assigned natural left outer join tainfo)
  select to_jsonb(coursesandtas.*) from coursesandtas;
  ```

to_jsonb
"{""ta"": {""name"": ""Ann"", ""email"": ""at12""}, ""cname"": ""AD"", ""semester"": 2}"
"{""ta"": {""name"": ""Ann"", ""email"": ""at12""}, ""cname"": ""OOP"", ""semester"": 2}"
"{""ta"": {""name"": ""Bob"", ""email"": ""bm13""}, ""cname"": ""OOP"", ""semester"": 2}"
"{""ta"": null, ""cname"": ""SDRP"", ""semester"": 7}"

OOP is represented twice because it has two TAs
  ```

Courses and TAs

- ```
  with
  tainfo as (select tid, jsonb_build_object('name',
             taname, 'email', email) as ta from tas),
  coursesandtas as
        (select cname, semester, jsonb_agg(ta) as ta
         from relcourses natural left outer join
              assigned natural left outer join tainfo
         group by cname, semester)
  select to_jsonb(coursesandtas.*) from coursesandtas;
  ```

to_jsonb
"{""ta"": [null], ""cname"": ""SDRP"", ""semester"": 7}"
"{""ta"": [{""name"": ""Ann"", ""email"": ""at12""}, {""name"": ""Bob"", ""email"": ""bm13""}], ""cname"": ""OOP"", ""semester"": 2}"
"{""ta"": [{""name"": ""Ann"", ""email"": ""at12""}], ""cname"": ""AD"", ""semester"": 2}"

OOP is now represented once! But we want a single object for all courses
  ```

Making a single document

```
with
tainfo as (select tid, jsonb_build_object('name',
           taname, 'email', email) as ta from tas),
coursesandtas as
      (select cname, semester, jsonb_agg(ta) as ta
       from relcourses natural left outer join
            assigned natural left outer join tainfo
       group by cname, semester),
jsonparts as (select to_jsonb(coursesandtas.*) as json
              from coursesandtas),
jsonarray as (select jsonb_agg(json) as jsonarray from
              jsonparts)
select jsonb_build_object('courses', jsonarray)
from jsonarray;
```

Getting ”pretty” JSON

```
select jsonb_pretty(jsonb_build_object(…as before…))
```

jsonb_pretty

```
"{
    ""courses"": [
        {
            ""ta"": [
                null
            ],
            ""cname"": ""SDRP"",
            ""semester"": 7
        },
        {
            ""ta"": [
                {
                    ""name"": ""Ann"",
                    ""email"": ""at12""
                },
                {
                    ""name"": ""Bob"",
                    ""email"": ""bm13""
                }
            ],
            ""cname"": ""OOP"",
            ""semester"": 2
        },
        {
            ""ta"": [
                {
                    ""name"": ""Ann"",
                    ""email"": ""at12""
                }
            ],
            ""cname"": ""AD"",
            ""semester"": 2
        }
    ]
}"
```

”No TA” should not be represented by ”[null]”. We fix that in the exercises

Summary

- JSON basics
- Storing and querying JSON in PostgreSQL
- SQL/JSON path expressions
- From tables to JSON

- Exercises on Moodle