Working with JSON in YSQL

Bryn Llewellyn

Friday, 3-June-2022

YFTT YugabyteDB Friday Tech Talks

yugabyteDB

What is JSON?

- Stands for JavaScript Object Notation
- A format for the serialization of hierarchically structured data — defined in RFC 7159
- Described just as a “data interchange format” for transporting data from one richly typed system to another — no operations specified
- Usually called a document because its representation is ordinary Unicode text
- Can represent values of four primitive data types: *string*, *number*, *boolean*, and *null*
- And of two compound data types: *object* (key-value pairs) and *array* (of values)
- Each object value, and each array value, can be primitive or compound — indefinitely deeply

Yugabyte © 2022 – All Rights Reserved

JSON in NoSQL and SQL

- Specified and first used in 2001
- Soon used as the key-value payload by various NoSQL databases (e.g. MongoDB)
- Programming languages brought libraries for constructing JSON documents, for extracting values at specified paths, concatenating, comparing, and editing JSON documents, and so on
- PostgreSQL brought JSON support in Version 9.2 (September 2012) with the (plain) *json* and *jsonb* datatypes—and functions and operators for their values
- Implements the same general JSON functionality that client-side programming languages and MongoDB do — but with its own operator syntax and function names

Yugabyte © 2022 – All Rights Reserved

SQL > NoSQL> Distributed SQL… so why is JSON relevant in YSQL?

1970 — Codd's seminal paper *"A Relational Model of Data for Large Shared Banks"*

1979 — Commercially RDBMSs first become available

2005 — The NoSQL *"key-value"* fashion is born (Google Bigtable & later Apache Cassandra,...)

2009 — MongoDB supports JSON operations to let keys have rich compound values

2012 — Google Spanner acknowledges that key-JSON pairs are a poor substitute for Codd's time-honored paradigm by marrying SQL with NoSQL's plumbing

2019 — YugabyteDB puts PostgreSQL's *actual* upper half on top of NoSQL's plumbing

2022 — Yugabyte, Inc. is thriving…

and YSQL *invevitably* supports all of Postgres's JSON features…

but doesn't the advent of Distributed SQL make this impressive fact irrelevant?

References

- JSON data types and functionality in the YugabyteDB YSQL doc

  covers what the Postgres doc covers

  but with many more self-contained, runnabe, examples

- JSON functions and operators

  the subsection describes about thirty-five of these

- my blog post on: blog.yugabyte.com

Yugabyte © 2022 – All Rights Reserved

01-examples-of-json-values-and-value-extraction.sql

```
::jsonb (from text —  implicit)
::text  (from jsonb — explicit)
the -> operator
```

Yugabyte © 2022 – All Rights Reserved

Use *jsonb* — don't use (plain) *json*

- YSQL has two data types for JSON values
  for use in table columns, in SQL, in PL/pgSQL, …
  - (plain) *json* holds the text value — but checks that it is well-formed
  - *jsonb* holds a parsed representation of the document's tree
- Of course, it's *slightly* quicker to typecast *text* to (plain) *json* than to typecast *text* to *jsonb*
- But all subsequent operations are quicker on *jsonb* values than on (plain) *json* values

Yugabyte © 2022 – All Rights Reserved

02-typecasting-text-to-json(b)-to-text.sql

```
::jsonb (from text  – explicit)
::text  (from jsonb – explicit)
jsonb_pretty()
```

Yugabyte © 2022 – All Rights Reserved

About *'{"x": 42, "y": null}'::jsonb* and *'{"x": 42}'::jsonb* — BEWARE!

- The two object values:
  - `j1 := '{"x": 42, "y": null}'::jsonb`
  - `j2 := '{"x": 42}'::jsonb`

  have the same semantics w.r.t. value extraction so that, here:

  - `j1->>'y' is null  AND  j2->>'y' is null`

- But, surprisingly:
  - `j1 != j2`

- This can really bite you if you don't explicitly acknowledge this in some tests.

Yugabyte © 2022 – All Rights Reserved

03-jsonb-null-semantics.sql

::jsonb (from *text* – explicit)

::text (from *jsonb* – explicit)

jsonb_populate_record()

to_jsonb()

the ->> operator

Yugabyte © 2022 – All Rights Reserved

```
table j_books(
  k ... primary key,
  book_info jsonb not null)
```

Populating It

Creating Indexes and Constraints

Testing the Constraints

Yugabyte © 2022 – All Rights Reserved

04-create-and-populate-j-books-table.sql

05-alter-j-books-add-indexes-and-constraints.sql

06-do-manual-constraint-violation-tests.sql

the -> and ->> operators

jsonb_typeof()

jsonb_object_keys()

jsonb_array_length()

Yugabyte © 2022 – All Rights Reserved

Ad hoc JSON Queries

Yugabyte © 2022 – All Rights Reserved

07-query-the-j-books-table.sql

the -> and ->> operators

the ? operator

the @> operator

jsonb_array_elements()

CROSS JOIN LATERAL *and* WITH ORDINALITY

PG doc: https://www.postgresql.org/docs/current/queries-table-expressions.html

Laurenz Albe: https://www.cybertec-postgresql.com/en/cross-join-in-postgresql/

Yugabyte © 2022 – All Rights Reserved

Two Representations of the Same Information: JSON and "Classic" Relational

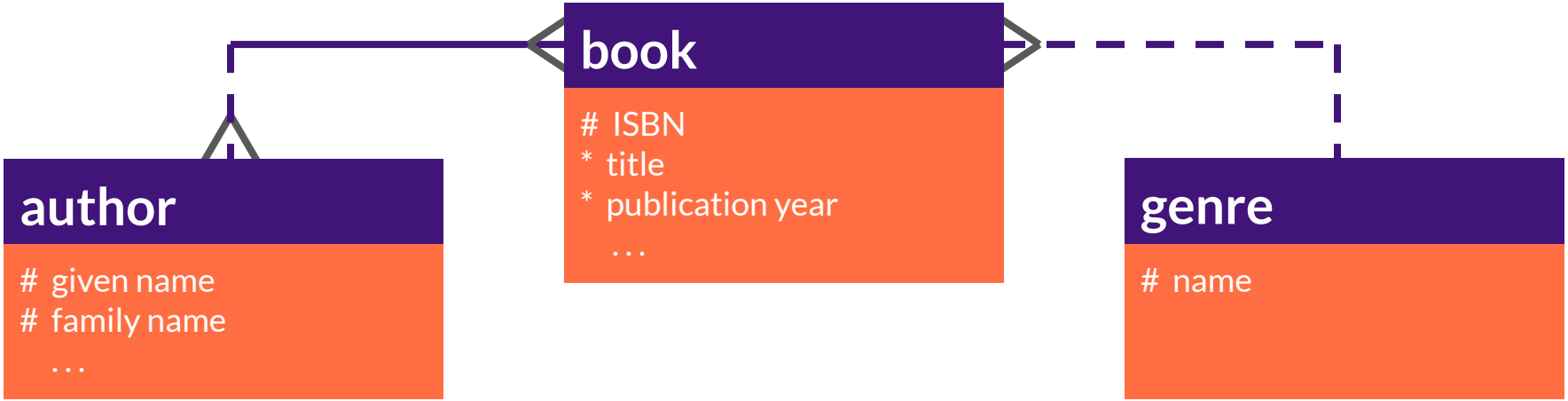
Transforming from JSON to Relational

Transforming from Relational to JSON

Proving that JSON and Relational Representations are Semantically Identical

Yugabyte © 2022 – All Rights Reserved

Deduce the relational model by eyeballing the eight JSON documents



- *Each* **book** *must have at least one* **author.**
  Each **author** *may be among the authors of one or several* **books.**

- *Each* **book** *may be of exactly one (known)* **genre.**
  Each **genre** *may classify one or several* **books.**

Yugabyte © 2022 – All Rights Reserved

08-create-j-books-r-view-and-populate-r-books.sql

09-create-r-books-j-view.sql

10-assert-j-books-r-books-j-view-identical.sql

the -> and ->> operators

`jsonb_typeof()`

`jsonb_populate_record()`

`jsonb_array_length()`

`to_jsonb()`

Yugabyte © 2022 – All Rights Reserved

Which representation meets your needs?

- If the incoming JSON adheres to a stable schema, then you'll probably want to "shred" it into a classic relational representation.
- If:
  - the incoming JSON's schema changes periodically, or
  - a typical incoming document uses only some of the attributes that the schema allows
- then you'll probably want the *DML-time* flexibility that comes from a table with a PK and a *jsonb* payload.
- You might want a hybrid approach: several classical columns with appropriate SQL data types (and esp. FKs to reference table rows) together with a *jsonb* "flex-field" column.

Yugabyte © 2022 – All Rights Reserved

Bonus: Do Try This at Home!

Hide the Internals of Your App's Database Module Behind an Impenetrable PL/pgSQL API

Only JSON Gives the Flexibility You Need to Return Expected Results or an Explanation of an Expected User-Error

Yugabyte © 2022 – All Rights Reserved

Demo

```
select insert_master_and_details_status(
  '{"m": "John", "ds": ["kettle", "pitcher", "saucepan"]}'::text);
```

→ *{"status": "success"}*

```
select insert_master_and_details_status(
  '{"m": "John", "ds": ["knife", "fork", "saucepan"]}'::text);
```

→ *{"reason": "Existing master \"John\" already has detail \"saucepan\"", "status": "user error"}*

```
select master_and_details_report('{"key": "John"}'::text);
```

→ *{"status": "success", "m_and_ds": {"m": "John", "ds": ["kettle", "pitcher", "saucepan"]}}*

```
select master_and_details_report('{"key": "Bill"}'::text);
```

→ *{"reason": "The master business key \"Bill\" doesn't exist", "status": "user error"}*

Yugabyte © 2022 – All Rights Reserved

Demo – *cont.*

```
select master_and_details_report('{"ket": "Fred"}'::text);
```

→ *{"status": "unexpected error", "ticket": 42}*

```
ticket                | 42
unit                  | function function master_and_details_report(text)
returned_sqlstate     | 22004
column_name           |
constraint_name       |
pg_datatype_name      |
message_text          | null value cannot be assigned to variable "mv_in" declared NOT NULL
table_name            |
schema_name           |
pg_exception_detail   |
pg_exception_hint     |
pg_exception_context  | PL/pgSQL function master_and_details_report(text) line 15 during statement
                        block local variable initialization
```


Yugabyte © 2022 – All Rights Reserved

Thank You

Join us on Slack: yugabyte.com/slack (#yftt channel)

Star us on Github: github.com/yugabyte/yugabyte-db

YFTT YugabyteDB Friday Tech Talks

yugabyteDB