

Getting to grips with JSON in the Database



Niall Mc Phillips - Long Acre sàrl
niall.mcphillips@longacre.ch
[@Niall_McP](https://twitter.com/Niall_McP)



1



2



3

← 🪂 *Many Years of Cloud Experience*
Cloud Early Adopter ☁️

Three images of skydivers. The top-left image shows a solo skydiver in a red helmet. The bottom-left image shows a circular formation of skydivers. The right image shows a large group of skydivers in a complex formation.



4

About me: Niall Mc Phillips

Owner - Long Acre sàrl
Co-founder and Director - Stephenson and Associates (founded 1995)
Irish 🇮🇪 / 🇨🇭 Swiss Living in Geneva, Switzerland.

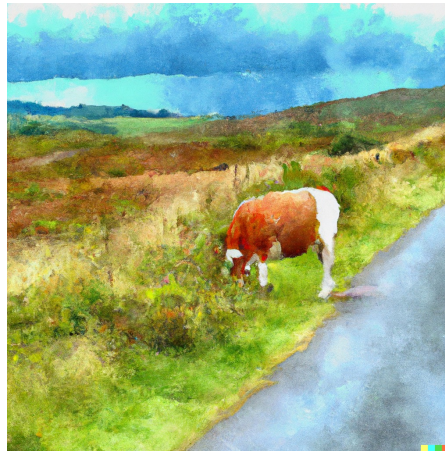


- Oracle ACE Pro ♠
- Symposium42 member
- Using Oracle database as a Developer and DBA for >30 years
- Developing web applications with Oracle DB since 1995
- Developing with APEX since 2005
- Organizer of the original Swiss APEX Meetup group


 @Niall_McP
 niall.mcphillips@longacre.ch



5






6



Oracle ACE Program




500+ technical experts helping peers globally


The **Oracle ACE Program** recognizes and rewards community members for their technical and community contributions to the

 Oracle ACE Director |
  Oracle ACE Pro |
  Oracle ACE Associate

ace.oracle.com

ace.oracle.com/nominate

Connect:  aceprogram_ww@oracle.com  Facebook.com/OracleACEs  [@oracleace](https://twitter.com/oracleace)



7

SYMPOSIUM 42

Created by the community, to support the community

Sharing of reliable knowledge
Supporting the various user groups and individuals

 [@sym_42](https://twitter.com/sym_42)

 <https://sym42.org/>

8

Relational – very-condensed history

- 1970 - First defined by E.F.Codd of IBM and was published in the IBM Systems Journal
- 1979 - a start-up company called “*Relational Software Inc.*” (RSI) released a product that they named “*Oracle*”
Interesting factoid, the first Oracle release was “version 2” – because no one would want to buy version 1



9

Relational - Normalisation

Let's start with a list of data representing short-term apartment rentals

Apartments							
Address	Description	Landlord	Landlord phone	Landlord e-mail	Currency	Price / week	Amenities
21 Rue du Saut	blah, blah	D. Jepp	022 678 4322	d.jepp@ap t.ch	CHF	980	Wifi Kitchen Balcony
62 Rue du Pirate	blah, blah	D. Jepp	022 678 4322	d.jepp@ap t.ch	CHF	1480	Wifi Kitchen Garden
42 Rue des Caraïbes	blah, blah	M. Curphy	01 78 43 22 56	m.curphy @xyz.ch	CHF	520	Wifi Kitchenette

10

Relational – 1st Normal Form

Multiple values not allowed in columns

Apartments							
Address	Description	Landlord	Landlord phone	Landlord e-mail	Currency	Price / week	Amenities
21 Rue du Saut	blah, blah	D. Jepp	022 678 4322	d.jepp@apt.ch	CHF	980	Wifi, Kitchen, Balcony
62 Rue du Pirate	blah, blah	D. Jepp	022 678 4322	d.jepp@apt.ch	CHF	1480	Wifi, Kitchen, Garden
42 Rue des Caraïbes	blah, blah	M. Curphy	01 78 43 22 56	m.curphy@xyz.ch	CHF	520	Wifi

11

Relational – 1st Normal Form

Multiple values not allowed in columns

Apartments							
Address	Description	Landlord	Landlord phone	Landlord e-mail	Currency	Price / week	Amenity
21 Rue du Saut	blah, blah	D. Jepp	022 678 4322	d.jepp@apt.ch	C	21 Rue du Saut	Wifi
62 Rue du Pirate	blah, blah	D. Jepp	022 678 4322	d.jepp@apt.ch	C	21 Rue du Saut	Kitchen
42 Rue des Caraïbes	blah, blah	M. Curphy	01 78 43 22 56	m.curphy@xyz.ch	C	21 Rue du Saut	Balcony
						62 Rue du Pirate	Wifi
						62 Rue du Pirate	Kitchen
						62 Rue du Pirate	Garden
						42 Rue des Caraïbes	Wifi

12

Relational – 2nd Normal Form

2nd Normal Form can be achieved by adding a single-value primary key

Apartments							
ID	Address	Description	Landlord	Landlord phone	Landlord e-mail	Currency	Price / week
1	21 Rue du Saut	blah, blah	D. Jepp	022 678 4322	d.jepp@apt.ch		
2	62 Rue du Pirate	blah, blah	D. Jepp	022 678 4322	d.jepp@apt.ch		
3	42 Rue des Caraïbes	blah, blah	M. Curphy	01 78 43 22 56	m.curphy@xyz.ch		

ID	Address	Amenity
1	21 Rue du Saut	Wifi
2	21 Rue du Saut	Kitchen
3	21 Rue du Saut	Balcony
4	62 Rue du Pirate	Wifi
5	62 Rue du Pirate	Kitchen
6	62 Rue du Pirate	Garden

13

Relational – 3rd Normal Form

Remove redundancies

Apartments						
ID	Address	Description	Landlord ID	Currency	Price / week	
1	21 Rue du Saut	blah, blah	1	CHF	980	
2	62 Rue du Pirate	blah, blah	1	CHF		
3	42 Rue des Caraïbes	blah, blah	2	CHF		

Landlords			
ID	Name	Phone	e-mail
1	D. Jepp	022 678 4322	d.jepp@apt.ch
2	M. Curphy	01 78 43 22 56	m.curphy@xyz.ch

Apartment Amenities	
Apartment ID	Amenity ID
1	1
1	2
1	3
2	1
2	2
2	4

Amenities	
ID	Amenity
1	Wifi
2	Kitchen
3	Balcony
4	Garden

14

Relational – Normalisation

We could now construct SQL statements joining tables to answer questions such as :

- Which apartments have kitchens and how much are they?
- Which apartments are operated by D. Jepp and what are their amenities?
- etc.

15

Relational – Major Benefits

- **Data Integrity** is ensured
- **Structure** is explicitly defined outside of the data
- **Reliability**, tried and tested approaches
- **Easily-defined transactions**

16

Relational – Some Drawbacks

- **Lack of flexibility**
- **Potential for Complexity**

17

JSON

- **Dates from the early-2000's by Douglas Crockford**
- **First standardized in 2013 (*ECMA-404*)**
- **2017 – ISO/IEC standard (*ISO/IEC 21778:2017*)**
- **Independent of underlying technologies**
- **Wide adoption in the development community**

18

JSON

Let's take a look at our short-stay apartment list

19

A JSON object for one apartment

```
{
  "id":1,
  "address":"21 Rue du Saut",
  "description":"blah, blah",
  "weeklyPrice":"980",
  "currency":"CHF",
  "landlord":{"name":"D.Jepp",
             "phone":"022 678 4322",
             "email":"d.jepp@apt.ch"},
  "amenities":["Wifi",
             "Kitchen",
             "Balcony"]
}
```

20

JSON

«Great - But...»

let's look at this in a different way

21

A JSON object for one landlord

```
{
  "id": "1",
  "name": "D. Jepp",
  "phone": "022 678 4322",
  "email": "d.jepp@apt.ch",
  "apartments": [
    {
      "id": "1",
      "address": "21 Rue du Saut",
      "description": "blah, blah",
      "weeklyPrice": "980",
      "currency": "CHF",
      "amenities": ["Wifi",
                   "Kitchen",
                   "Balcony"]
    },
    {
      "id": "2",
      "address": "62 Rue du Pirate",
      "description": "blah, blah",
      "weeklyPrice": "1480",
      "currency": "CHF",
      "amenities": ["Wifi",
                   "Kitchen",
                   "Garden"]
    }
  ]
}
```

22

Adding reviews

Let's add reviews from people that have stayed in the apartments

- Reviewer ID
- Reviewer Name
- Stars Given
- Review Text

23

Adding reviews - Relational

Data Model changes – add at least 2 tables

- a table of Reviewers with ID and Name
- a table of Reviews

Reviews			
APT ID	Reviewer ID	Stars	Review Text
1	1	5	Great apartment!
1	2	4	Nice apartment, but

Reviewers	
ID	Name
1	James Plunkett
2	James Connolly

24

Adding reviews - JSON

- Add an array of reviews

```
{
  "id":1,
  "address":"21 Rue du Saut",
  ...
  ,
  "reviews":[{"reviewerId":1,
              "name":"James Plunkett",
              "stars":5,
              "text":"Great apartment!"},
            {"reviewerId":2,
              "name","James Connolly",
              "stars":4,
              "text":"Nice Apartment, but"}
          ]
}
```

25

End of philosophical discussion – enough for now

Now let's get our hands on the good stuff...

38

What we're going to look at now

- Defining a JSON column in a table

39

What we're going to look at now

- Defining a JSON column in a table
- **Inserting JSON in different ways**

40

What we're going to look at now

- Defining a JSON column in a table
- Inserting JSON in different ways
- **Querying JSON in multiple ways**
 - **Dot notation**

41

What we're going to look at now

- Defining a JSON column in a table
- Inserting JSON in different ways
- Querying JSON in multiple ways
 - Dot notation
 - **Projecting JSON as relational**

42

What we're going to look at now

- Defining a JSON column in a table
- Inserting JSON in different ways
- Querying JSON in multiple ways
 - Dot notation
 - Projecting JSON as relational
- **Updating JSON**

43

What we're going to look at now

- Defining a JSON column in a table
- Inserting JSON in different ways
- Querying JSON in multiple ways
 - Dot notation
 - Projecting JSON as relational
- Updating JSON
- **Indexing JSON**

44

Defining a JSON column – 19c

```
create table nobel_prizes
(year      number,
 category_id number,
 prize_details clob);

alter table nobel_prizes add constraint
ck_laureates_json
  check (prize_details IS JSON);
```

45

Defining a JSON column – 21c+

```
create table nobel_prizes
(year      number,
 category_id number,
 prize_details json);
```

46

Inserting JSON

```
insert into nobel_prizes (year, category_id, prize_details)
values (2022,7,'{ "year": "2022",
                "categoryId": "7",
                "categoryName": "APEX",
                "details":
[ {"id" : "2000",
  "firstname" : "Developer",
  "surname" : "Community",
  "motivation" : "for the development of great applications",
  "shareFraction" : "1" }
]
}');
```

47

Inserting JSON using JSON_OBJECT

```
insert into nobel_prizes (year, category_id, prize_details)
select 2022, 7,
json_object
  (key 'year'      is 2022,
   key 'categoryId' is 7,
   key 'categoryName' is 'APEX',
   key 'details'   is
     json_arrayagg
       (json_object(key 'id'      is d.id,
                    key 'firstname' is d.firstname,
                    key 'surname'  is d.surname,
                    key 'motivation' is d.motivation,
                    key 'shareFraction' is d.shareFraction)
        returning clob)
  )
from ... d;
```

48

Demo time!

49

Querying JSON – Dot notation

```
select
  p.year,
  p.category_id,
  json_query(p.prize_details, '$.details'
    returning varchar2(4000) pretty) laureates
from nobel_prizes p;
```

50

Querying JSON – Dot notation

```
select p.prize_details.year,  
       p.prize_details.categoryName,  
       p.prize_details.details[0].firstname as firstname1,  
       p.prize_details.details[0].surname  as surname1,  
       p.prize_details.details[1].firstname as firstname2,  
       p.prize_details.details[1].surname  as surname2  
from nobel_prizes p;
```

51

Querying JSON – All array elements as a JSON array

```
select  
  p.prize_details.year.string() as year,  
  p.prize_details.categoryName.string()  
                                as category,  
  p.prize_details.details[*].surname  
                                as laureates  
from nobel_prizes p;
```

52

Querying JSON – Using JSON_TABLE to project as multiple rows

```
select p.prize_details.year.string() as year,  
       p.prize_details.categoryName.string() as category,  
       j.firstname || ' ' || j.surname as name,  
       j.share_fraction,  
       j.motivation as motivation  
from nobel_prizes p  
join categories c on (c.category_id = p.category_id)  
cross join json_table(p.prize_details, '$.details[*]'  
                     columns (firstname varchar2(30) path '$.firstname',  
                               surname   varchar2(30) path '$.surname',  
                               share_fraction number      path '$.shareFraction',  
                               motivation  varchar2(500) path '$.motivation')) j;
```

53

Updating JSON – JSON_TRANSFORM changing a value

```
update nobel_prizes p  
set p.prize_details  
  = json_transform  
    (p.prize_details,  
     set '$.categoryName' = 'APEKS')  
where p.year=2022  
and p.category_id = 7;
```

55

Updating JSON – JSON_TRANSFORM removing an array element

```
update nobel_prizes p
  set p.prize_details
    = json_transform
      (p.prize_details,
       remove '$.details[*]?(@.id=="2000")')
  where p.category_id = 7;
```

56

Updating JSON – JSON_TRANSFORM adding an array element

```
update nobel_prizes p
  set p.prize_details
    = json_transform (p.prize_details,
      append '$.details' =
      json_object (key 'id' is '2000',
                  key 'firstname' is 'Developer',
                  key 'surname' is 'Community',
                  key 'motivation' is great applications',
                  key 'shareFraction' is 2)
      )
  where p.category_id = 7;
```

57

Indexing JSON

- Function indexes for simple cases
- Multivalue indexes for array elements
- Search Index for other searches

See [Search indexes for JSON](#) – Roger Ford, Oracle - 23rd Nov 2021

59

Indexing JSON - Simple cases

- Function indexes for simple cases

```
create index ind_nobel_prizes$1
  on nobel_prizes
  (prize_details.year.string());
```

```
then
select * from nobel_prizes p
  where p.prize_details.year.string() = '1916';
```

60

Indexing JSON - Multivalue indexes

For array elements:

```
create multivalue index ind_apartments$2
  on nobel_prizes p
  (p.prize_details.details[*].firstname.string())
;
```

then...

```
select p.* from nobel_prizes p
  where json_exists(p.prize_details,
    '$.details?(@.firstname = "Annie")');
```

61

Indexing JSON – Search Indexes

For textual searches (similar to Oracle Text):

```
create search index ind_nobel_prizes$3
  on nobel_prizes (prize_details) for json;
```

then...

```
select p.* from nobel_prizes p
  where json_textcontains(p.prize_details,
    '$.details.motivation',
    'novel');
```

62

JSON in the DB Use Cases – some examples

- Equipment certification – the certificates should reflect only the certificate information issued at the date of issue despite any changes to the data structure since certification.
- Auditing – allows data changes to be tracked over an evolving data model
- Fast-moving, “temporary” data – i.e. this month's “special pick”

63

23C

64

23c - JSON Relational Duality

65

23c - JSON Relational Duality



66

23c - Creating a JSON Relational Duality view

What we are about to see

- 1) Creation of a JSON Relational Duality view of prizes and their laureates.
- 2) Creation of a JSON Relational Duality view of laureates and their prizes (i.e. a very different JSON to the previous one)
- 3) Updating of one of the views and viewing the updates via both views and directly on the relational tables.
- 4) ETAGs and their use for optimistic locking.

67

23c - Creating a JSON Relational Duality view

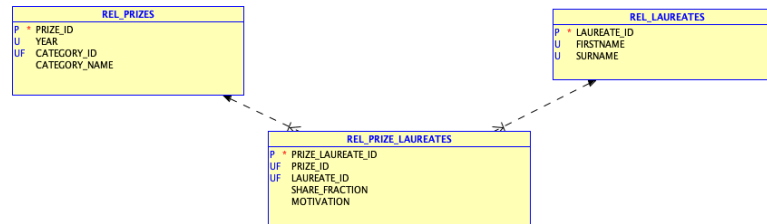
For this example, we have 3 relational tables

- REL_LAUREATES
- REL_PRIZE_LAUREATES
- REL_PRIZES

68

23c - Creating a JSON Relational Duality view

Simple model with primary keys and foreign keys.



69

23c - Creating a JSON Relational Duality view of Prizes and their laureates

```
create or replace json relational duality view dv_nobel_prizes as
select json {'prizeld' : p.prize_id,
           'year'   : p.year,
           'category' : p.category_name,
           'laureates' :
             [ select json {'prizeLaureateId' : pl.prize_laureate_id,
                           ...
             ]
from rel_prizes p with insert update delete;
```

70

23c - Creating a different JSON Relational Duality view of Laureates and their Prizes

```
create or replace json relational duality view dv_nobel_prizes2
as
select json {'laureateId' : l.laureate_id,
           'firstname' : l.firstname,
           'surname'   : l.surname,
           'prizes' :
             [ select json {'prizeLaureateId' : pl.prize_laureate_id,
                           ...
             rel_laureates l with insert update delete;
```

71

23c - Updating a JSON Relational Duality view

- JSON Updates
- Relational update
- ETAGs

72

Relational JSON Duality 23c – The DEMO



73

JSON in Oracle – Multiple avenues

- Oracle SODA – accepts JSON from multiple environments
 - *Java, Node.js, REST, C, Python, PL/SQL*
- Oracle's new MongoDB Drivers and Tools
- REST and ORDS
- SQL and PL/SQL

74

Advantages of Relational / JSON Hybrid models

- Less tables, more flexibility
- Very infrequently used attributes don't need to be modelled as stringently
- Modern approach that non-Oracle developers can quickly identify with and adopt

78

Challenges of Hybrid models

- ***With more flexibility - attention needs to be paid to ensuring data integrity.***

79

Challenges of Hybrid models

- With more flexibility - attention needs to be paid to ensuring data integrity.
- ***Finding the right balance between relational and JSON for your data, your application and your environment.***

80

Challenges of Hybrid models

- With more flexibility - attention needs to be paid to ensuring data integrity.
- Finding the right balance between relational and JSON for your data, your application and your environment.
- ***23c JSON Relational Duality has addressed these challenges.***

81

JSON in the Database

- JSON in the Database offers new opportunities and techniques for dealing with data
- JSON Relational Duality is a game changer
- JSON in the Database is here to stay
- Embrace it and add it to our toolkit

82



83