Mixing Relational with NoSQL
Heresy or Harmony?

Presented on 13th October 2022

at

HrOUG 2022

Rovinj, Croatia

by

Niall Mc Phillips - Long Acre sàrl

niall.mcphillips@longacre.ch

@Niall_McP

long acre
solutions today

1

2

3

4

About me: Niall Mc Phillips

Owner - Long Acre sàrl (founded 2015)
Co-founder and Director - Stephenson and Associates (founded 1995)
Irish 🇮🇪 / 🇨🇭 Swiss Living in Geneva, Switzerland.

- Oracle ACE — ORACLE ACE
- 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 Swiss APEX Meetup group

@NiallMcP
niall.mcphillips@longacre.ch

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

Oracle ACE

ace.oracle.com/nominate

ace.oracle.com

Connect: aceprogram_ww@oracle.com Facebook.com/OracleACEs @oracleace

Relational – very-condensed history

- 1970 - First defined by E.F.Codd of IBM and was published in the IBM Systems Journal

long acre
solutions today

7

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

long acre
solutions today

8

Relational - Normalisation

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

Apartments

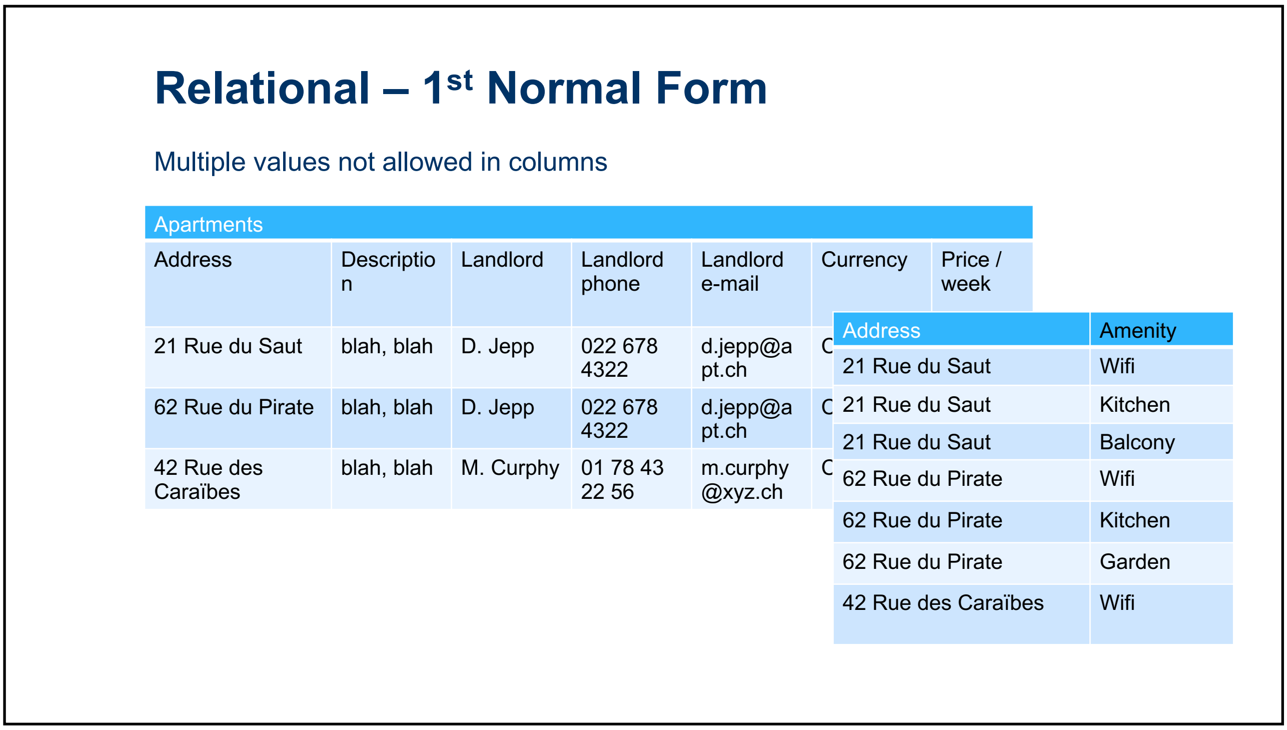
Address	Description	Landlord	Landlord phone	Landlord e-mail	Currency	Price / week	Amenities
21 Rue du Saut	blah, blah	D. Jepp	022 678 4322	d.jepp@apt.ch	CHF	980	Wifi Kitchen Balcony
62 Rue du Pirate	blah, blah	D. Jepp	022 678 4322	d.jepp@apt.ch	CHF	1480	Wifi Kitchen Garden
42 Rue des Caraïbes	blah, blah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

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*



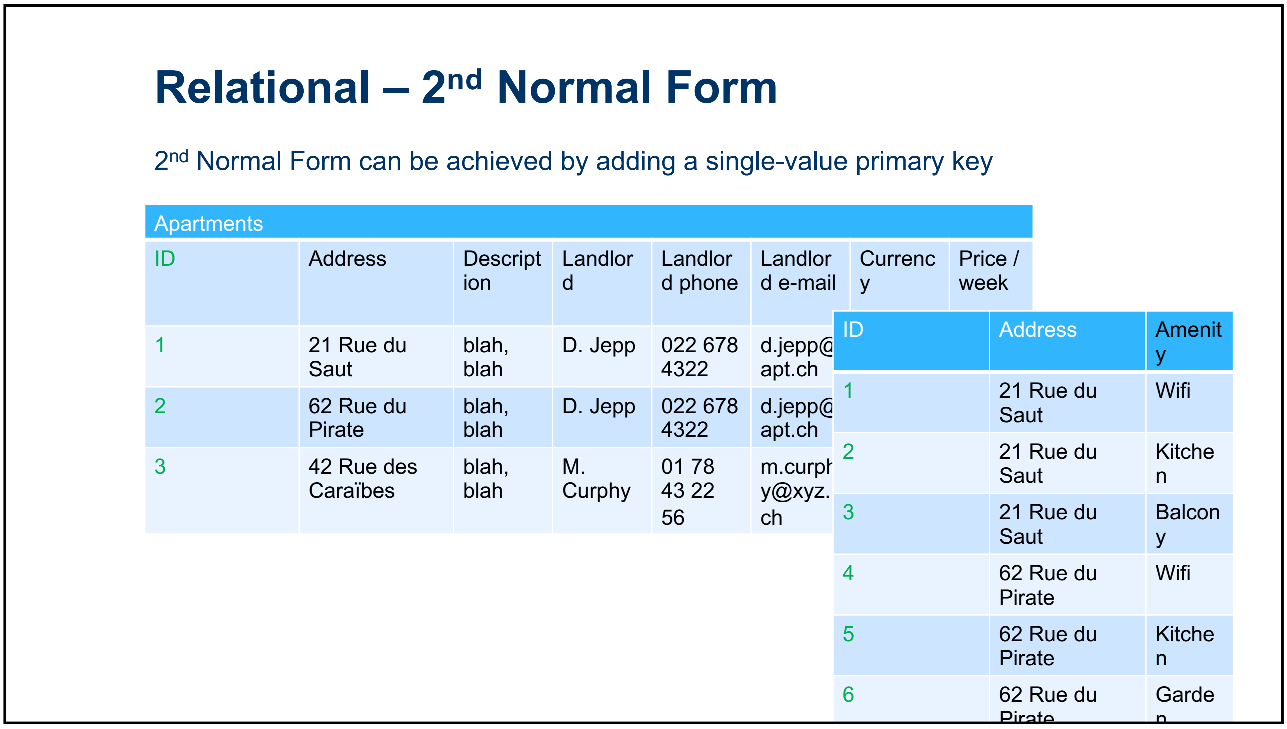
Relational – 1st Normal Form

Multiple values not allowed in columns

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

Address	Amenity
21 Rue du Saut	Wifi
21 Rue du Saut	Kitchen
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

11



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

12

Relational – 3rd Normal Form

Remove redundancies

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

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

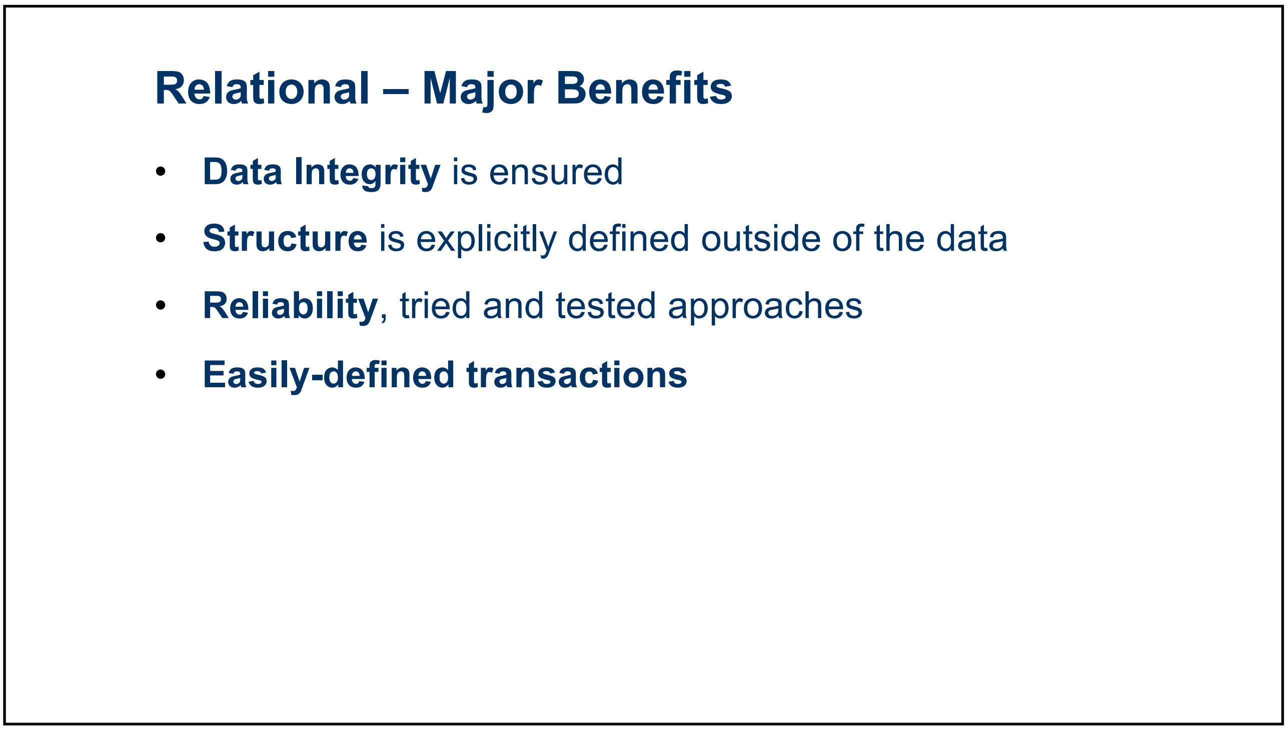
Apartment Amenities	
Apartment ID	Amenity ID
1	1
1	2
1	3
2	1
2	2
2	4

Amenities	
ID	Amenity
1	Wifi
2	Kitchen
3	Balcony
4	Garden

Relational – Normalisation

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

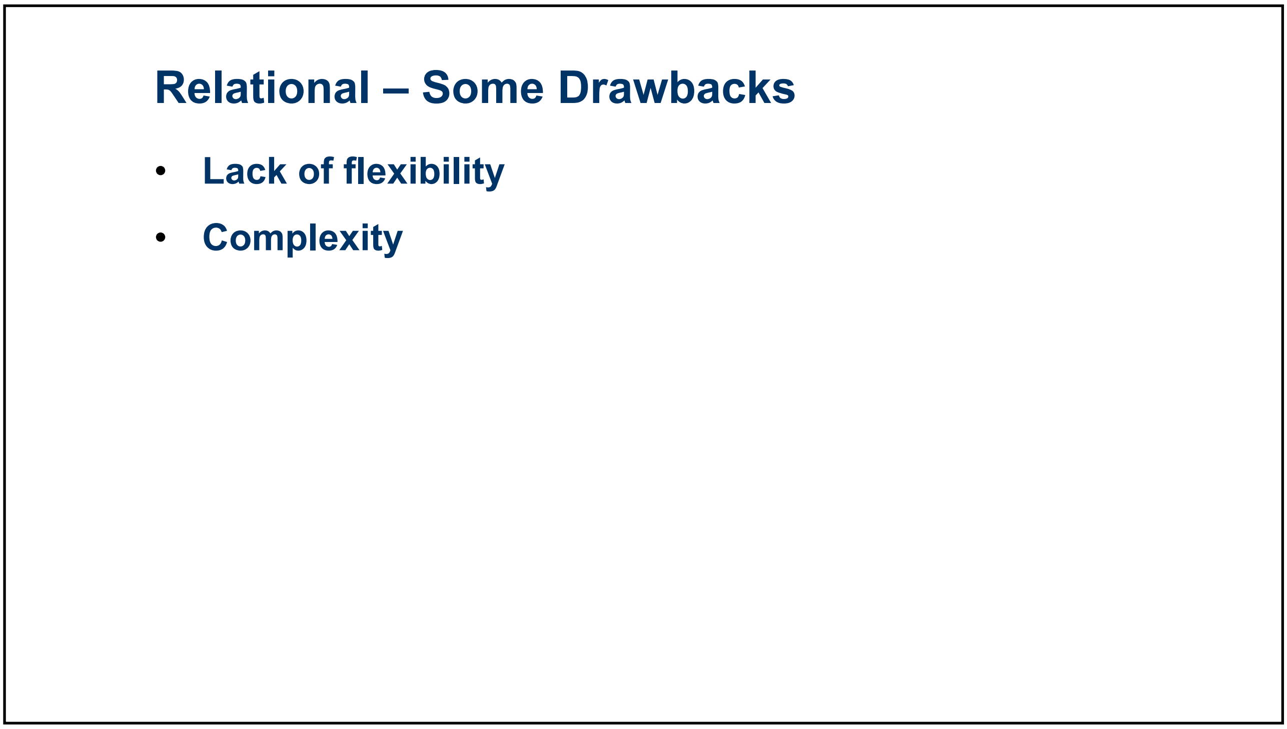
- Which apartments have kitchens and how much are they?
- Which apartments are operated by D. Jepp and what are their amenities?
- etc.



Relational – Major Benefits

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

15



Relational – Some Drawbacks

- **Lack of flexibility**
- **Complexity**

16

JSON

- **Dates from the early-2000's by Douglas Crockford**

17

JSON

- **Dates from the early-2000's by Douglas Crockford**
- **First standardized in 2013** *(ECMA-404)*

18

JSON

- **Dates from the early-2000's by Douglas Crockford**
- **First standardized in 2013** *(ECMA-404)*
- **2017 – ISO/IEC standard** *(ISO/IEC 21778:2017)*

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**

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

21

JSON

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

22

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"]
}
```

JSON

«Great - But…»

let's look at this in a different way

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"]
 }
]
}
```

Adding reviews

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

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

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

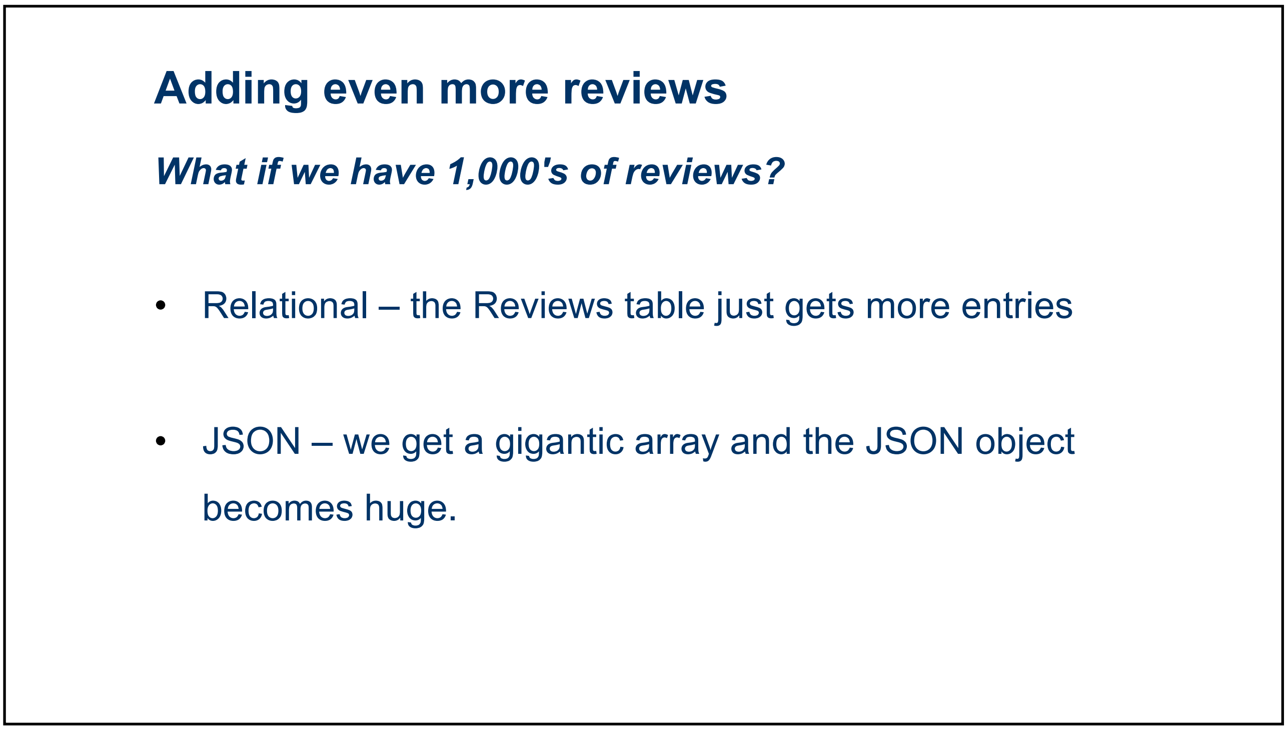
27

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"}
          ]
}
```

28



Adding even more reviews

What if we have 1,000's of reviews?

- Relational – the Reviews table just gets more entries
- JSON – we get a gigantic array and the JSON object becomes huge.

Adding even more reviews

What if we have 1,000's of reviews?

Adding even more reviews

What if we have 1,000's of reviews?

Another approach from the JSON viewpoint would be to predict application usage patterns.

31

Adding even more reviews

What if we have 1,000's of reviews?

Another approach from the JSON viewpoint would be to predict application usage patterns.

For example:

- ***Users usually view the apartment listing along with the description and the most recent reviews.***

32

Adding even more reviews

What if we have 1,000's of reviews?

Another approach from the JSON viewpoint would be to predict application usage patterns.

For example:

- *Users usually view the apartment listing along with the description and the most recent reviews.*
- *So, why not just keep the 5 most recent reviews in the JSON object?*

33

Once upon a time

Many, many years ago when I was learning about data modelling, one thing was repeatedly hammered into my brain

"Never consider the application

when modelling the data

- let the data speak for itself"

34

More recently

“Let's write the applications,

we'll structure the data as we evolve”

35

Culture Clash?

“Never consider the application

when modelling the data

- let the data speak for itself”

vs.

“Let's write the application,

we'll structure the data as we evolve”

36

Culture Clash?

Let see if there's another path...

What if we could do both?

- ***The benefits of the structure of a relational database for the core model***

PLUS

- ***The flexibility and ease of deployment of JSON***

JSON in the Oracle Database

With 21c we can now store JSON as a native datatype

JSON Binary Type

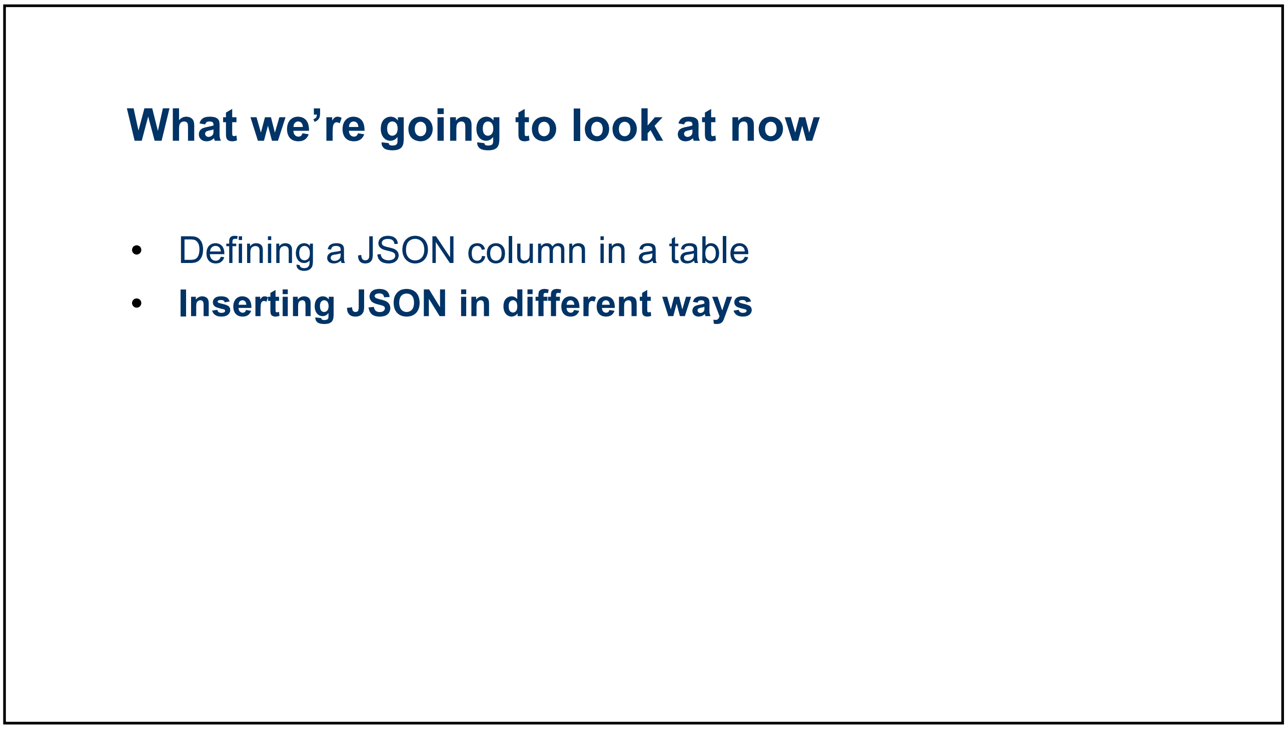
- New datatype in SQL, PL/SQL
- Natively supported in all drivers
  - OCI, JDBC, node.js, python (.NET in progress)
- Based on **OSON** - Optimized Binary representation
  - Self-contained format
  - Fast field lookups
  - Piecewise Updates possible
- Performance Benefits
  - Scans up to **5x** faster than textual JSON
  - Updates up to **10x** faster than textual JSON

What we’re going to look at now

What we’re going to look at now

- **Defining a JSON column in a table**

41



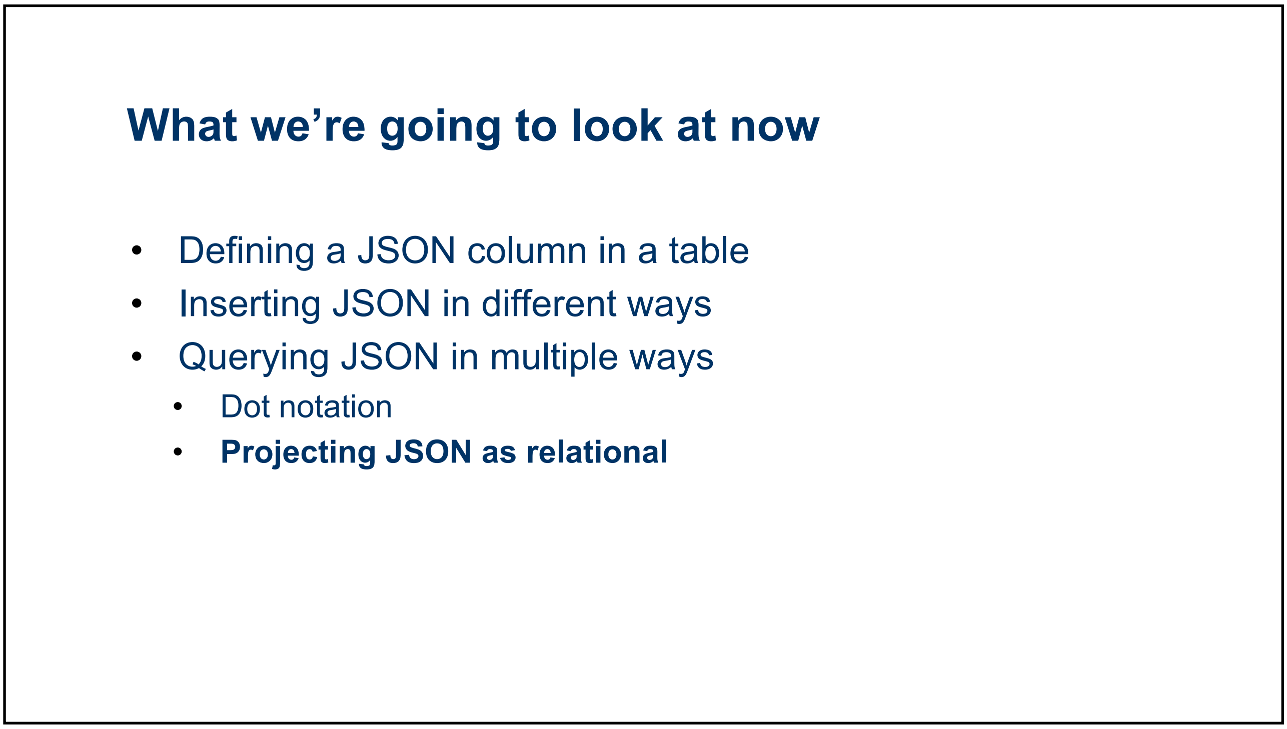
What we’re going to look at now

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

42

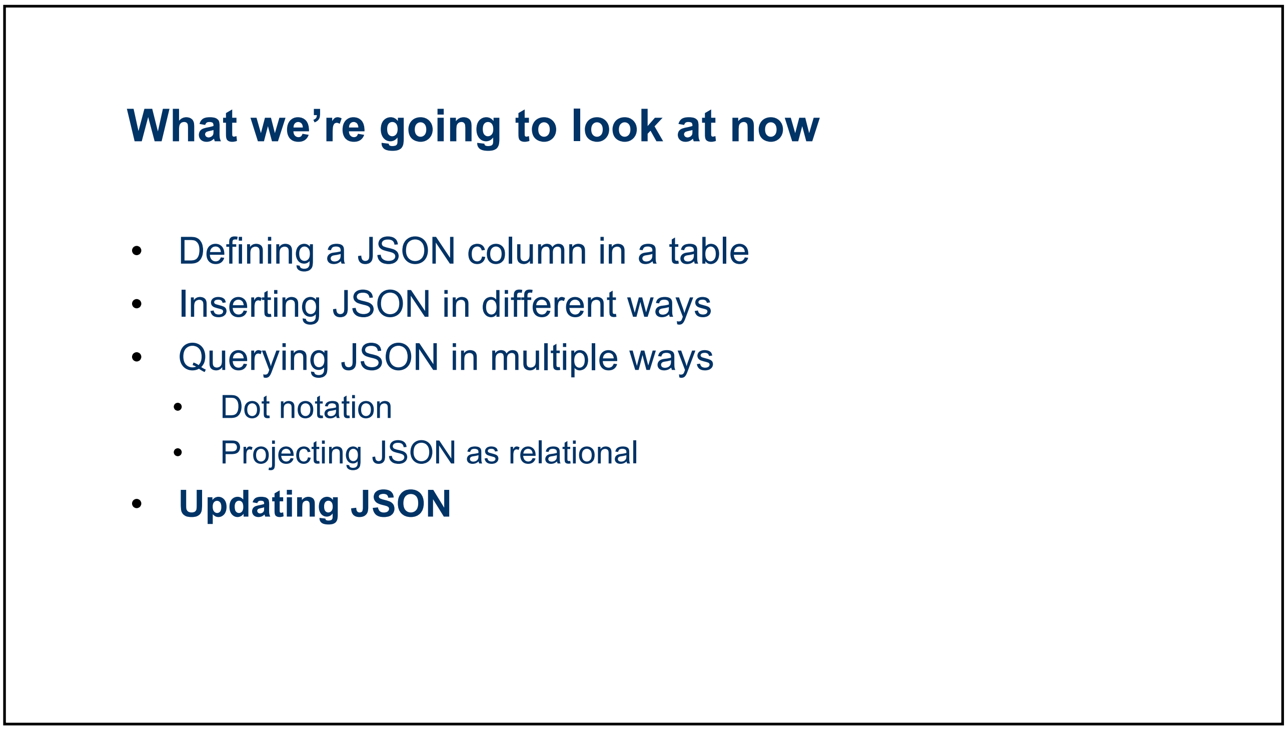
What we’re going to look at now

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



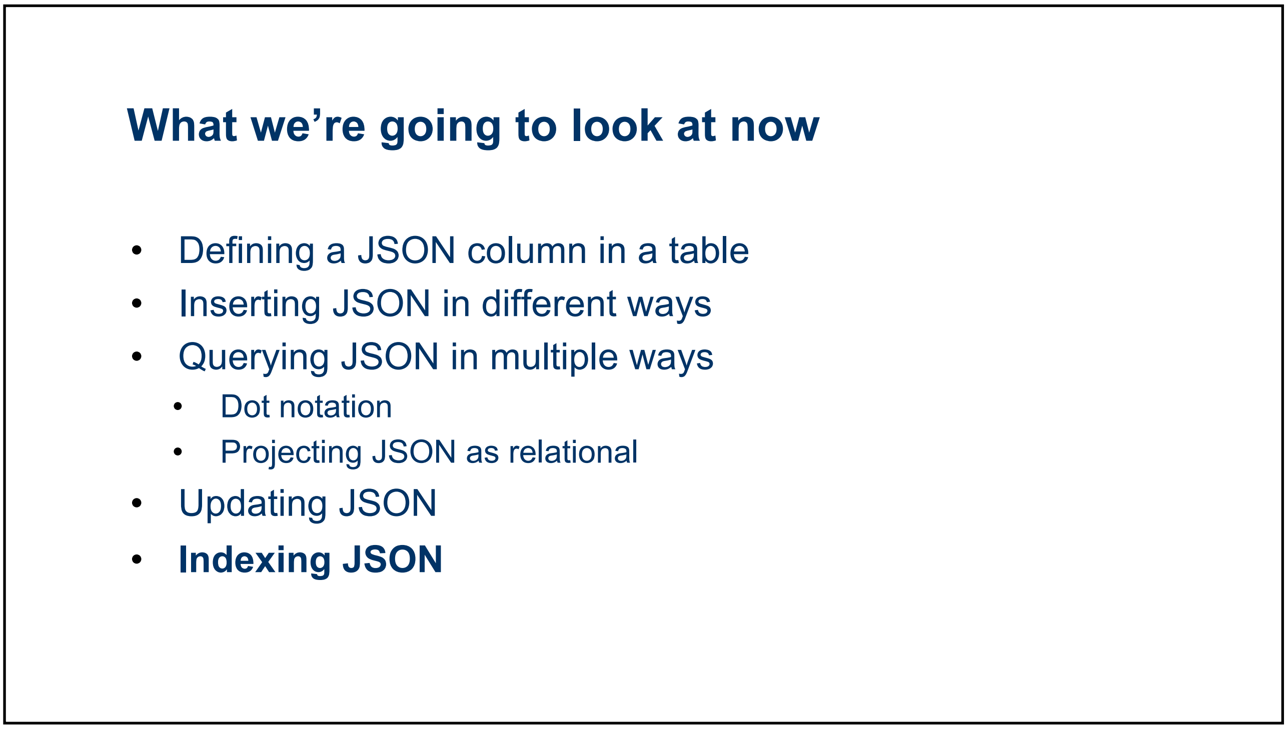
What we’re going to look at now

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



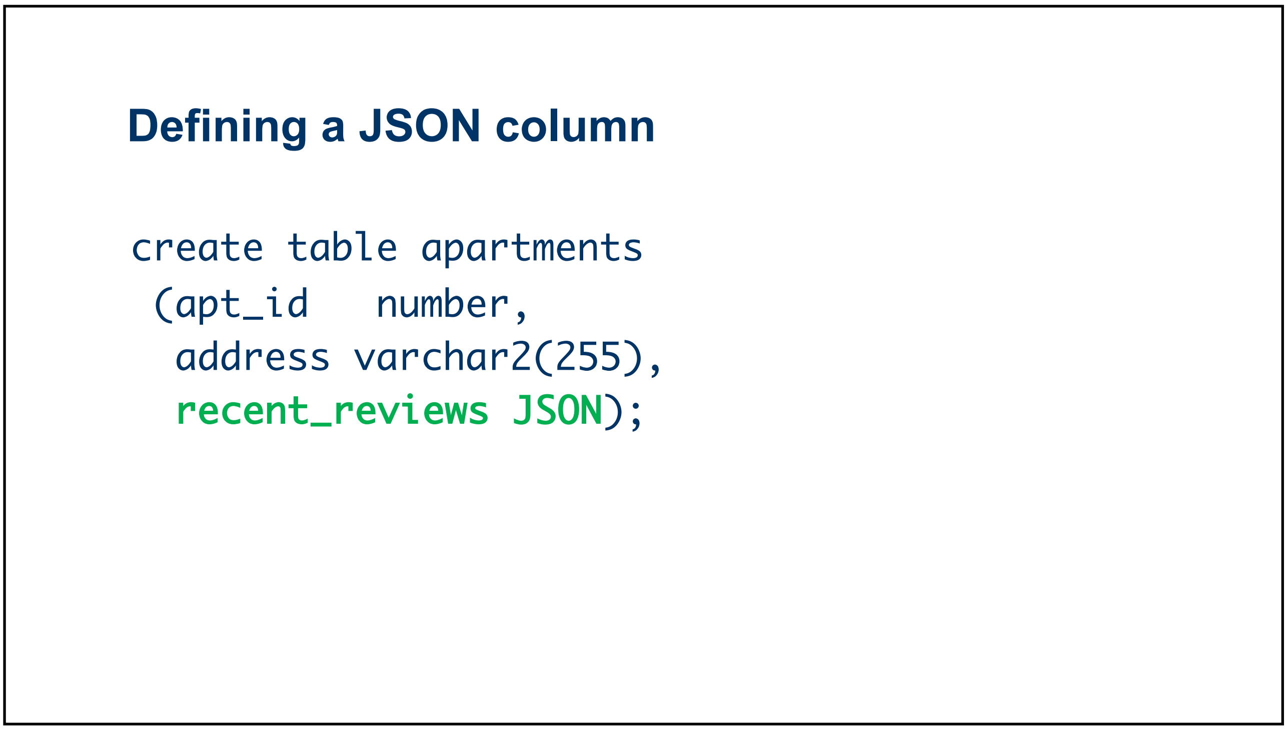
What we’re going to look at now

- 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**



What we’re going to look at now

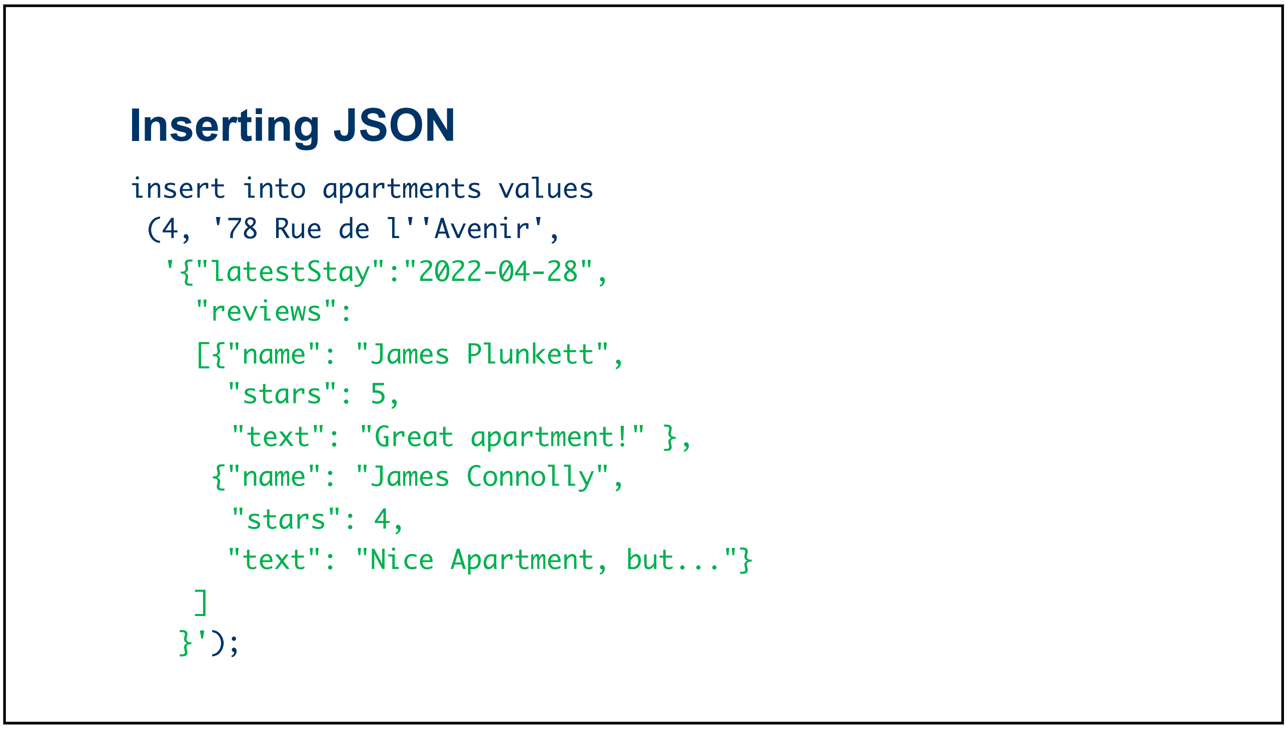
- Defining a JSON column in a table
- Inserting JSON in different ways
- Querying JSON in multiple ways
  - Dot notation
  - Projecting JSON as relational
- Updating JSON
- **Indexing JSON**



Defining a JSON column

```
create table apartments
 (apt_id   number,
  address varchar2(255),
  recent_reviews JSON);
```

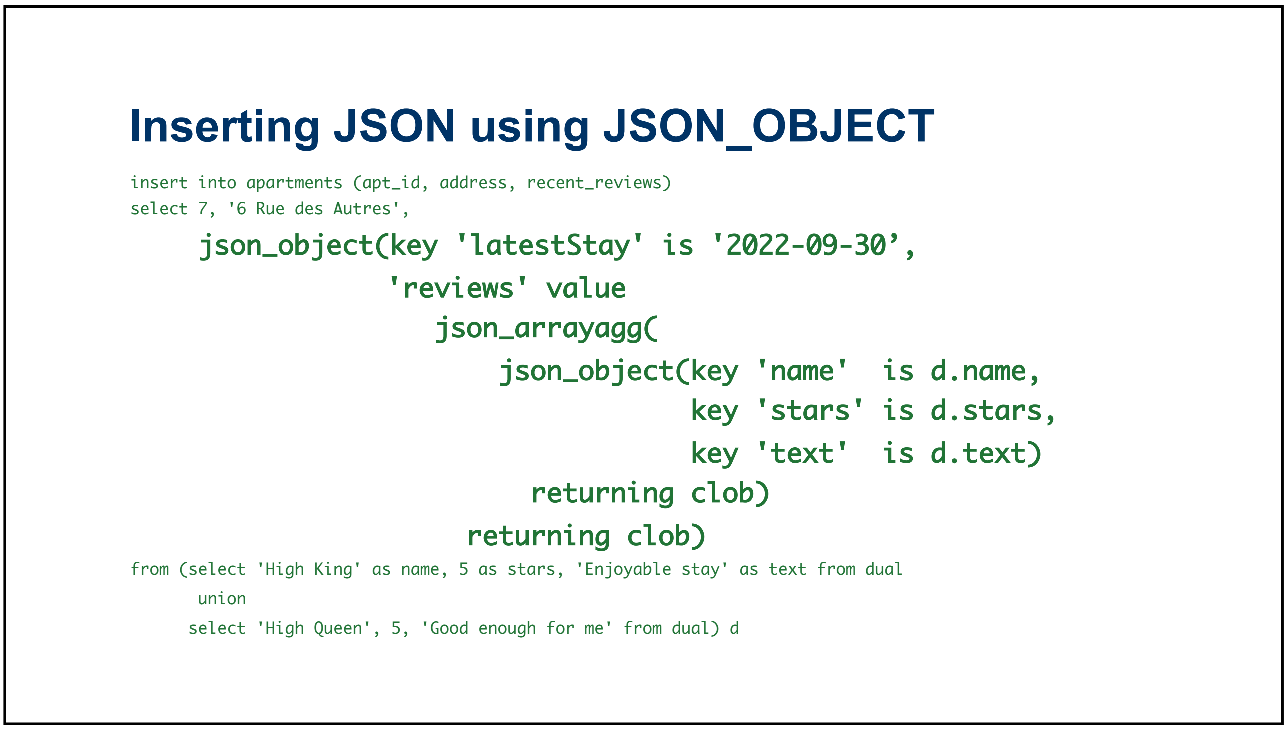
47



Inserting JSON

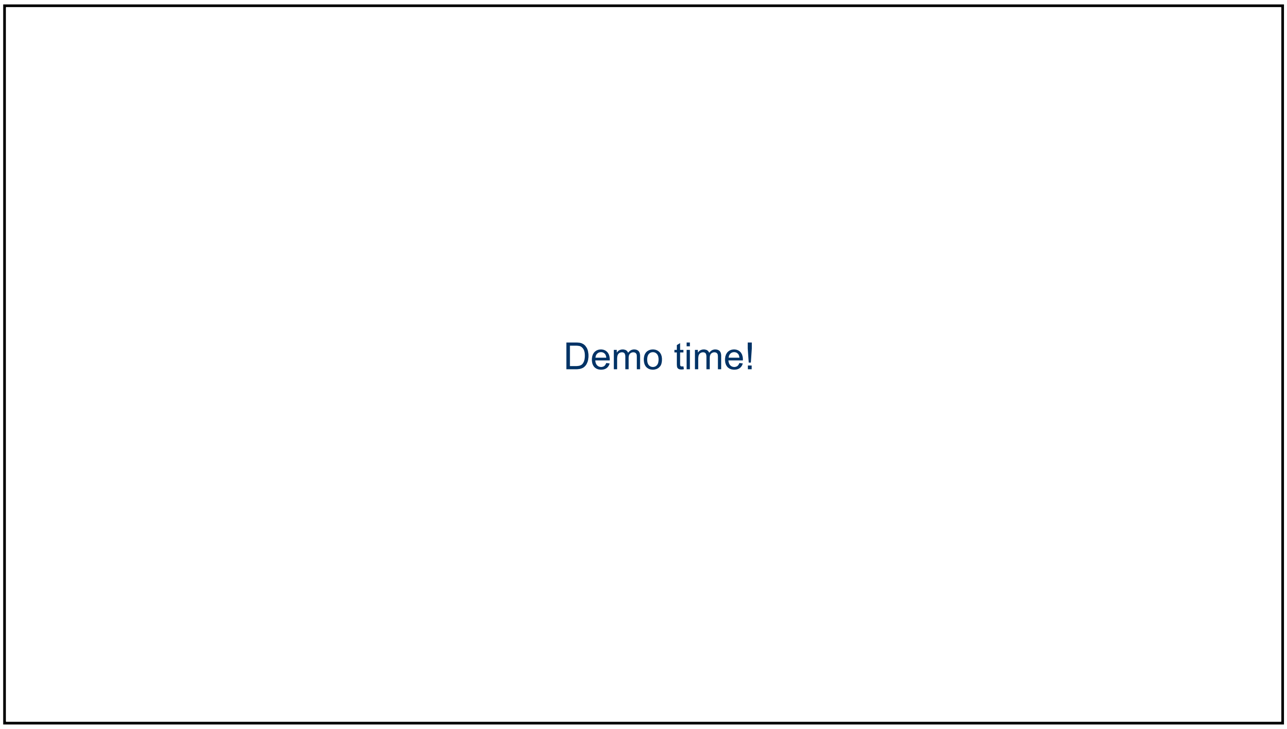
```
insert into apartments values
 (4, '78 Rue de l''Avenir',
  '{"latestStay":"2022-04-28",
    "reviews":
    [{"name": "James Plunkett",
      "stars": 5,
      "text": "Great apartment!" },
     {"name": "James Connolly",
      "stars": 4,
      "text": "Nice Apartment, but..."}
    ]
   }');
```

48

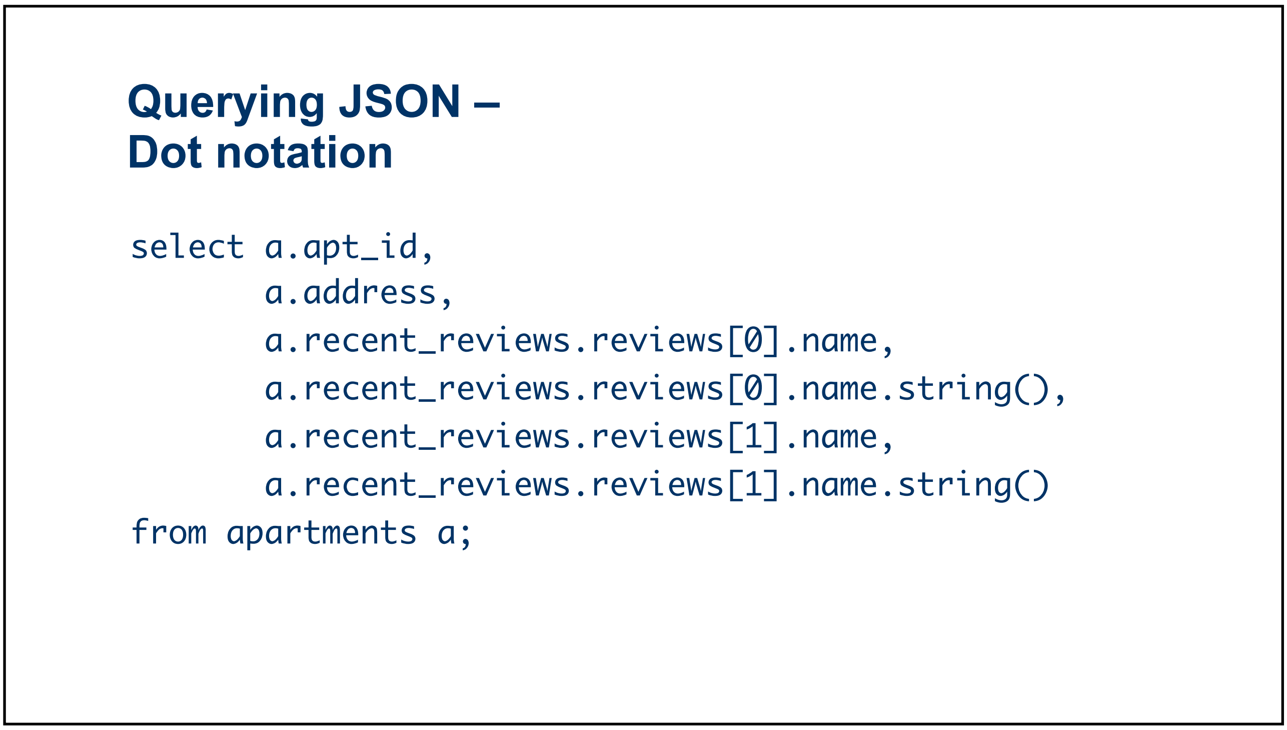


Inserting JSON using JSON_OBJECT

```
insert into apartments (apt_id, address, recent_reviews)
select 7, '6 Rue des Autres',
    json_object(key 'latestStay' is '2022-09-30’,
                'reviews' value
                   json_arrayagg(
                       json_object(key 'name'  is d.name,
                                   key 'stars' is d.stars,
                                   key 'text'  is d.text)
                         returning clob)
                     returning clob)
from (select 'High King' as name, 5 as stars, 'Enjoyable stay' as text from dual
       union
      select 'High Queen', 5, 'Good enough for me' from dual) d
```



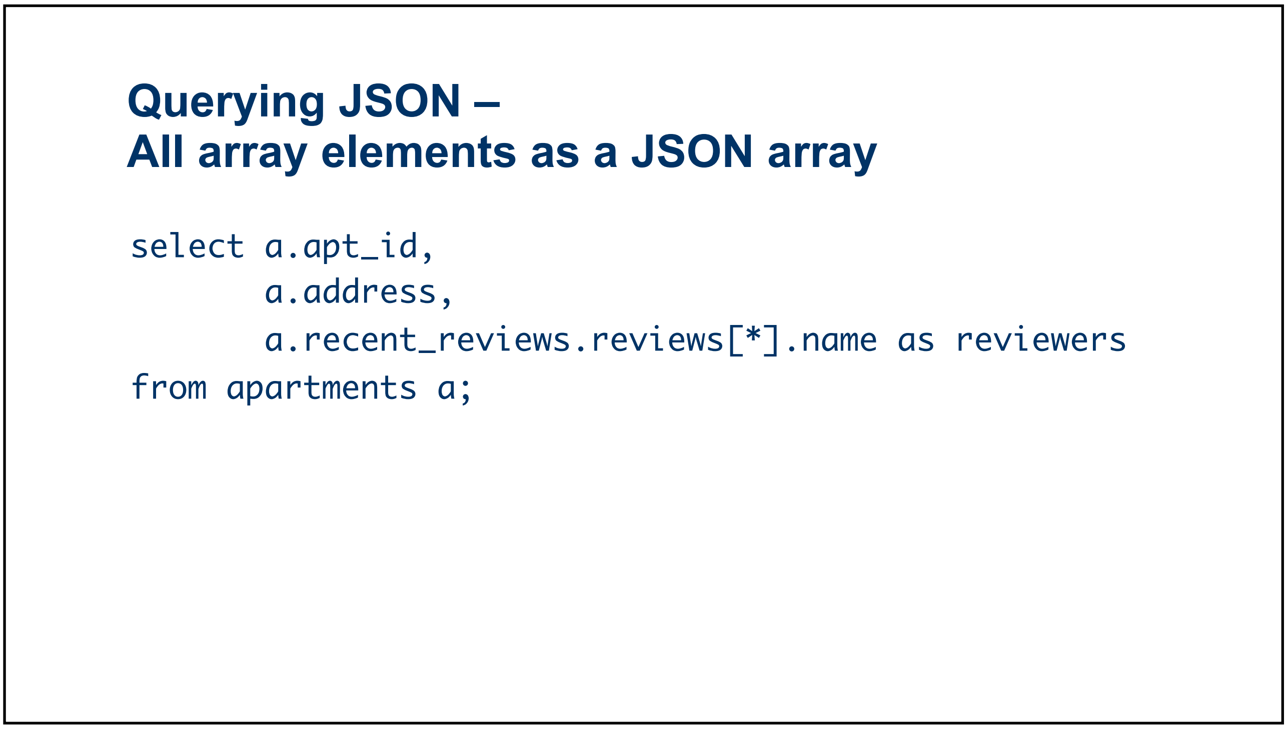
Demo time!



Querying JSON – Dot notation

```
select a.apt_id,
       a.address,
       a.recent_reviews.reviews[0].name,
       a.recent_reviews.reviews[0].name.string(),
       a.recent_reviews.reviews[1].name,
       a.recent_reviews.reviews[1].name.string()
from apartments a;
```

51



Querying JSON – All array elements as a JSON array

```
select a.apt_id,
       a.address,
       a.recent_reviews.reviews[*].name as reviewers
from apartments a;
```

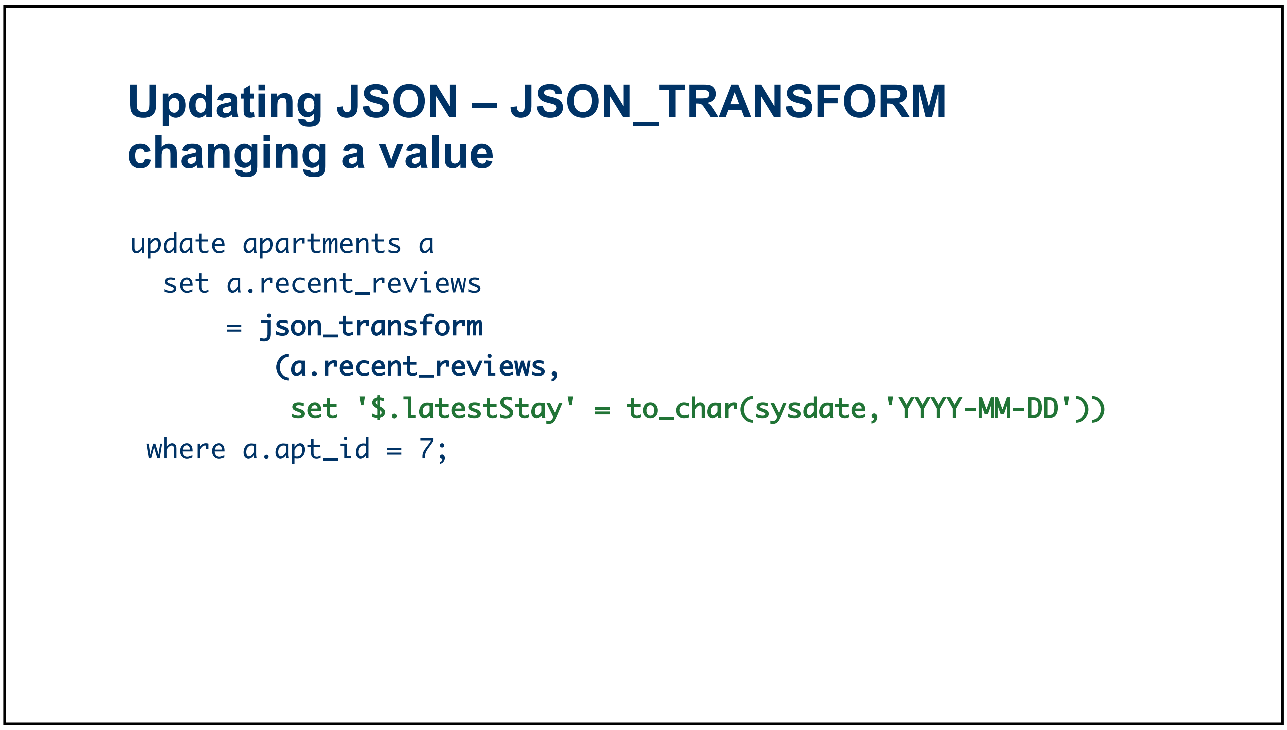
52

Querying JSON –
Using JSON_TABLE to return multiple rows

```
select a.apt_id, a.address,
       j.name as reviewer,
       j.stars,  j.text as review
  FROM apartments a,
       json_table(a.recent_reviews, '$.reviews[*]'
         columns (name  varchar2(30) path '$.name',
                  stars number       path '$.stars',
                  text  varchar2(50) path '$.text')) j;
```

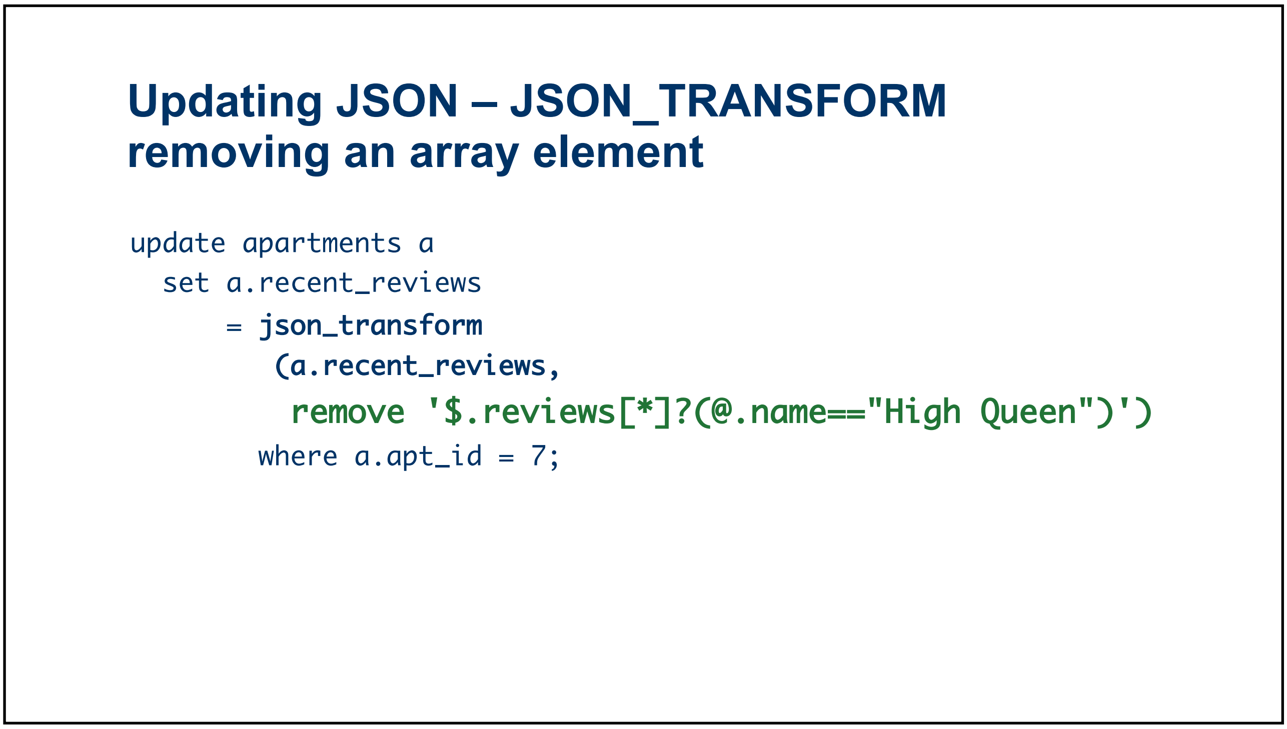
Querying JSON –
Creating a view on the JSON values

```
create or replace view vw_apartment_reviews as
select a.apt_id, a.address,
       j.name as reviewer,
       j.stars,  j.text as review
  FROM apartments a,
       json_table(a.recent_reviews, '$.reviews[*]'
         columns (name  varchar2(30) path '$.name',
                  stars number       path '$.stars',
                  text  varchar2(50) path '$.text')) j;
```



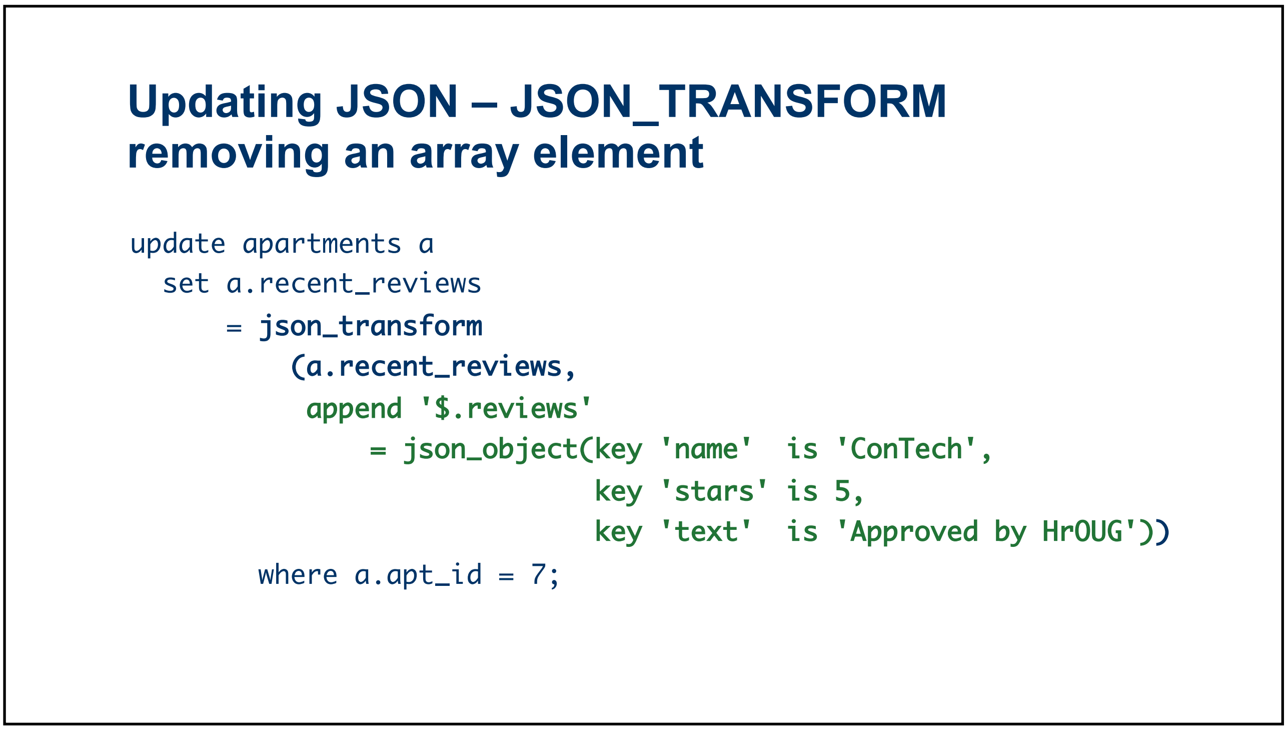
Updating JSON – JSON_TRANSFORM changing a value

```
update apartments a
  set a.recent_reviews
      = json_transform
         (a.recent_reviews,
          set '$.latestStay' = to_char(sysdate,'YYYY-MM-DD'))
 where a.apt_id = 7;
```



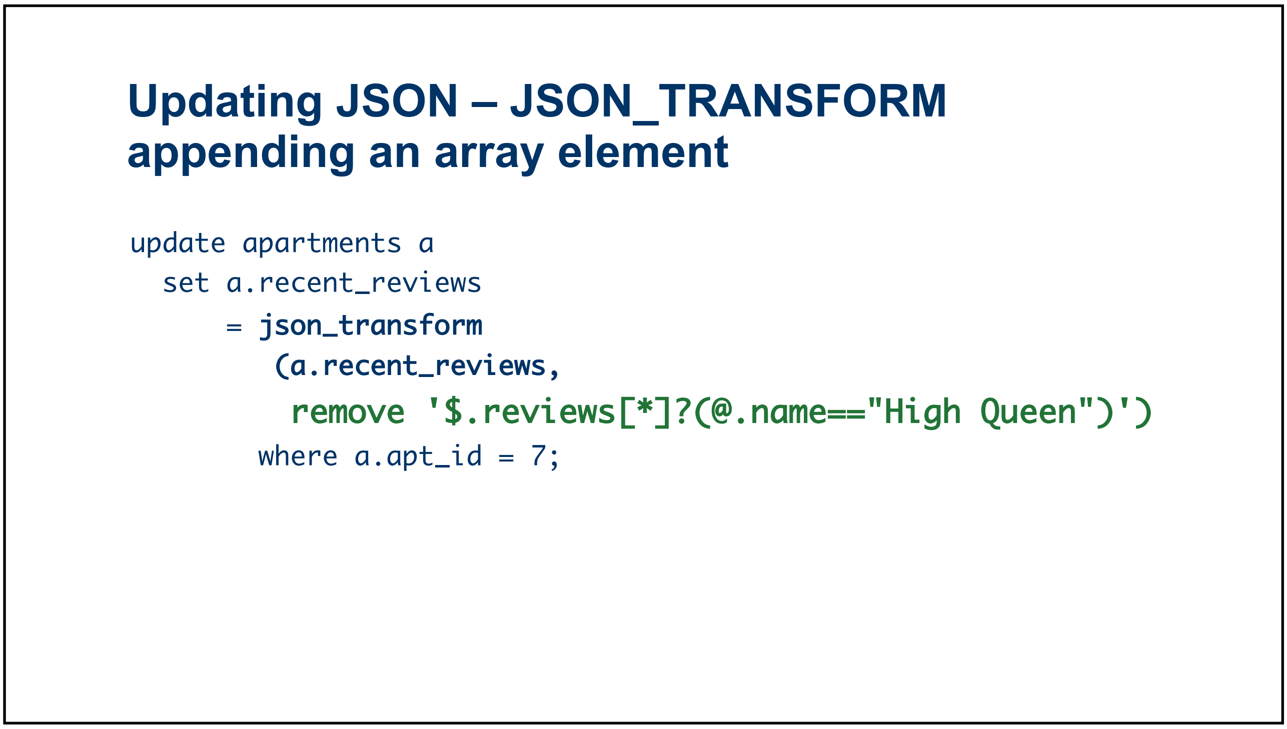
Updating JSON – JSON_TRANSFORM removing an array element

```
update apartments a
  set a.recent_reviews
      = json_transform
         (a.recent_reviews,
          remove '$.reviews[*]?(@.name=="High Queen")')
        where a.apt_id = 7;
```



Updating JSON – JSON_TRANSFORM removing an array element

```
update apartments a
  set a.recent_reviews
      = json_transform
          (a.recent_reviews,
           append '$.reviews'
               = json_object(key 'name'  is 'ConTech',
                             key 'stars' is 5,
                             key 'text'  is 'Approved by HrOUG'))
        where a.apt_id = 7;
```



Updating JSON – JSON_TRANSFORM appending an array element

```
update apartments a
  set a.recent_reviews
      = json_transform
         (a.recent_reviews,
          remove '$.reviews[*]?(@.name=="High Queen")')
        where a.apt_id = 7;
```

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

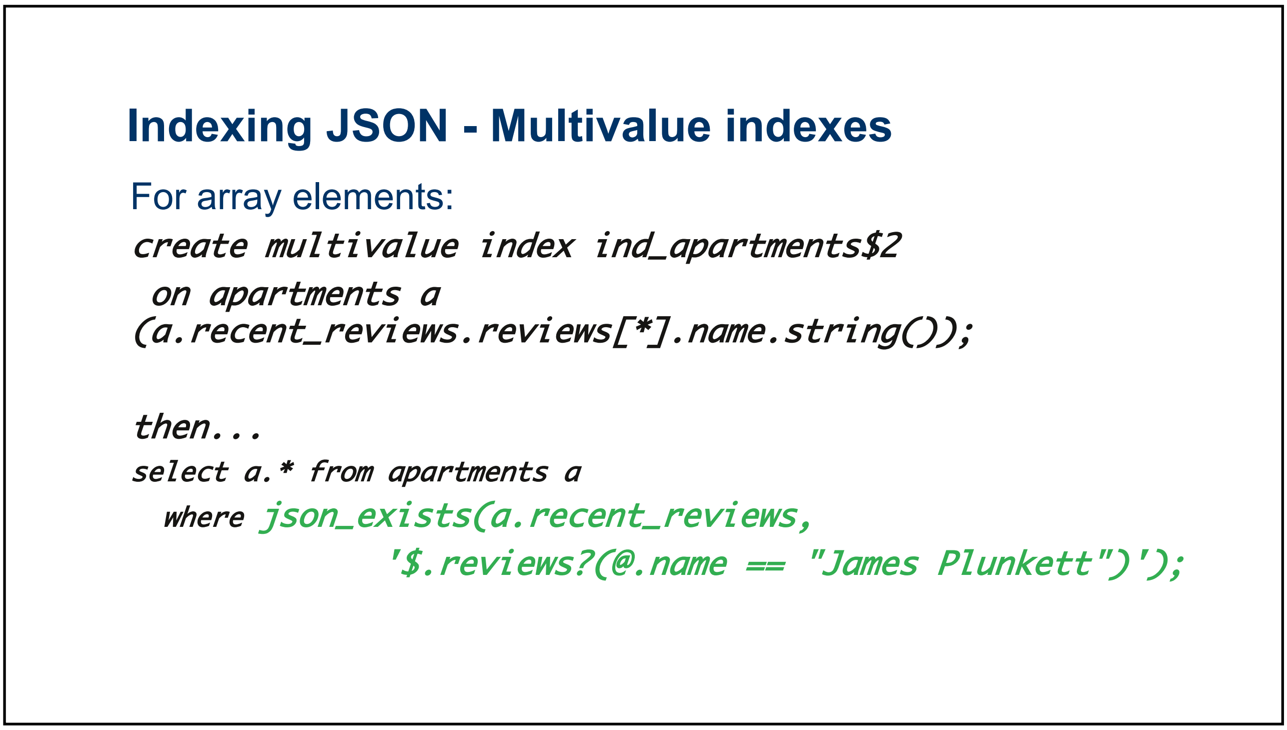
- Function indexes for simple cases

```
create index ind_apartments$1
    on apartments
(recent_reviews.latestStay.string());
```

then

```
select * from apartments a
where a.recent_reviews.latestStay.string() = '2022-09-30';
```

60



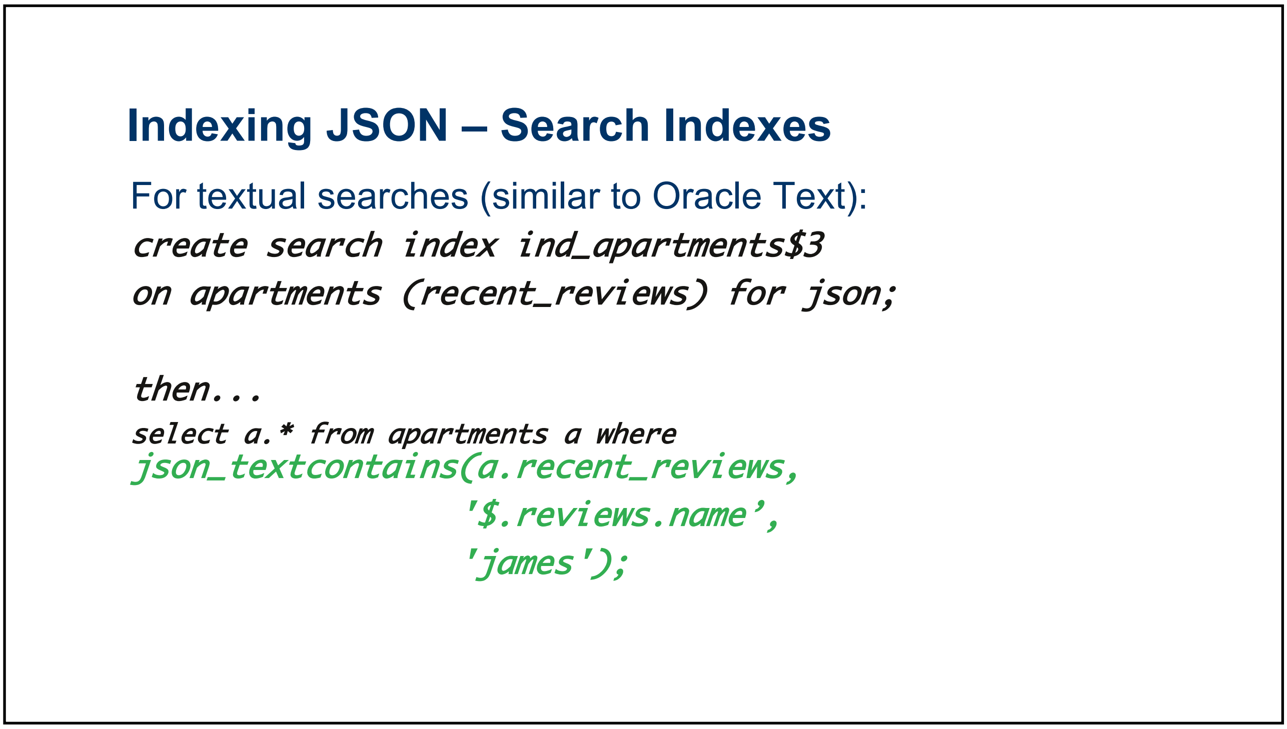
Indexing JSON - Multivalue indexes

For array elements:

```
create multivalue index ind_apartments$2
 on apartments a
(a.recent_reviews.reviews[*].name.string());

then...
select a.* from apartments a
  where json_exists(a.recent_reviews,
               '$.reviews?(@.name == "James Plunkett")');
```

61



Indexing JSON – Search Indexes

For textual searches (similar to Oracle Text):

```
create search index ind_apartments$3
on apartments (recent_reviews) for json;

then...
select a.* from apartments a where
json_textcontains(a.recent_reviews,
                  '$.reviews.name’,
                  'james');
```

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”

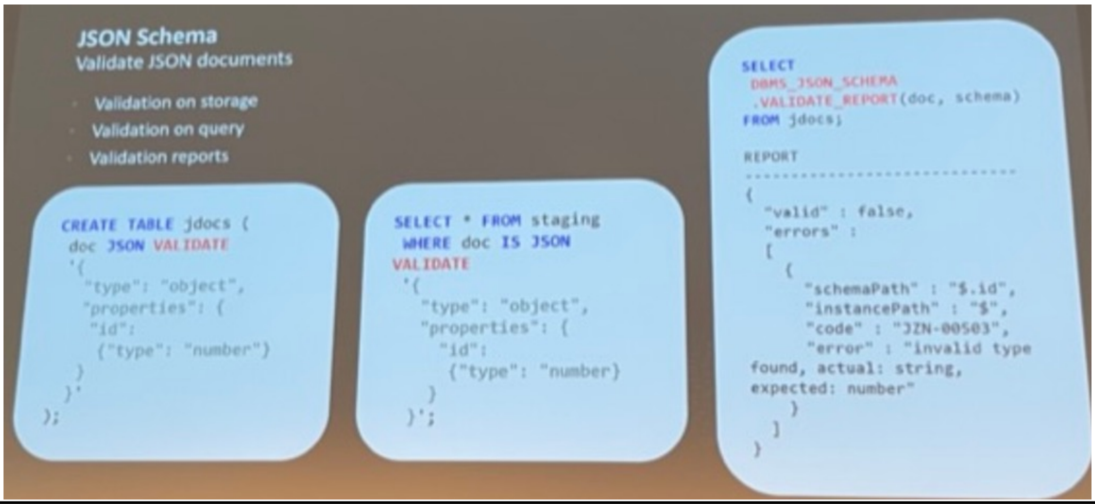
Upcoming Oracle 23c JSON features

JSON Schema

- various options to validate documents against a JSON Schema definition. The **JSON Schema** can be defined on a table column – almost as a check constraint. It can be used in a query to only select values that satisfy the schema and using package ***dbms_json_schema*** validation reports can be retrieved on specific values against a JSON Schema.

Upcoming Oracle 23c JSON features

JSON Schema



JSON Schema

Validate JSON documents

- Validation on storage
- Validation on query
- Validation reports

```
CREATE TABLE jdocs (
 doc JSON VALIDATE
 '{
  "type": "object",
  "properties": {
   "id":
    {"type": "number"}
  }
 }'
);
```

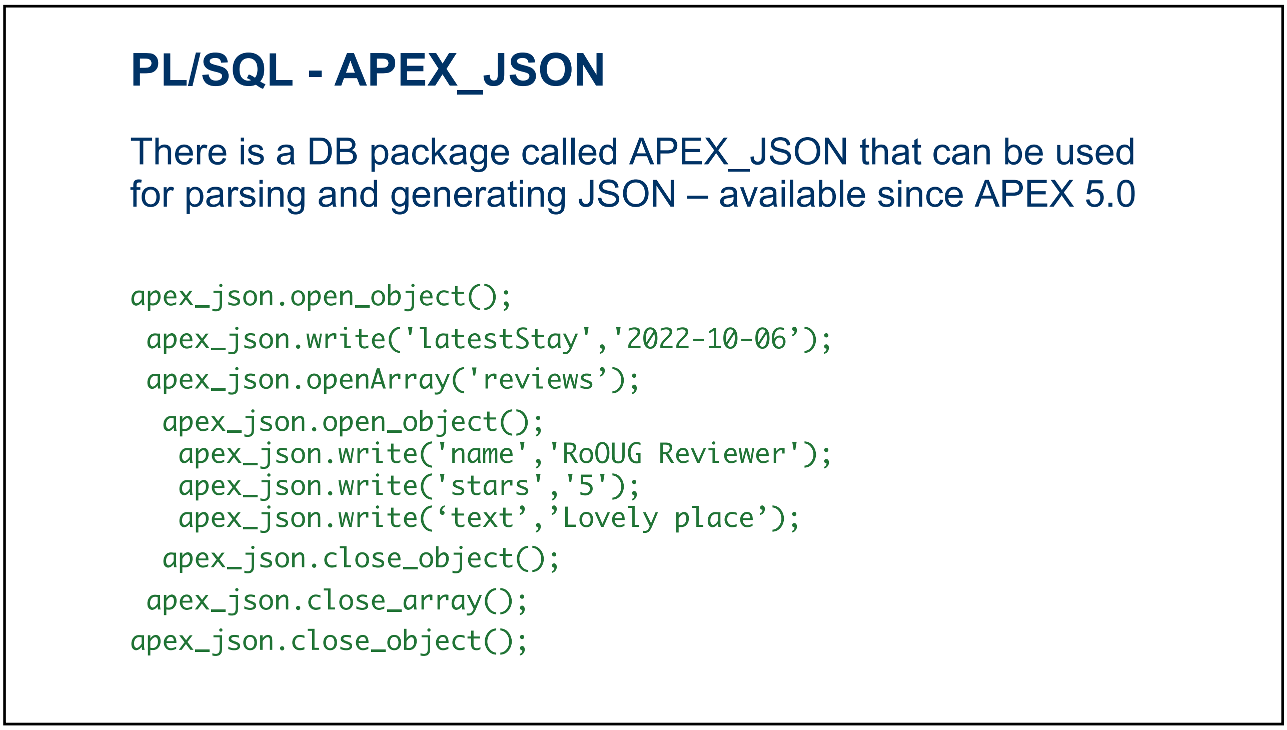
```
SELECT * FROM staging
 WHERE doc IS JSON
VALIDATE
 '{
   "type": "object",
   "properties": {
     "id":
      {"type": "number}
   }
 }';
```

```
SELECT
 DBMS_JSON_SCHEMA
 .VALIDATE_REPORT(doc, schema)
FROM jdocs;

REPORT
------------------------------
{
  "valid" : false,
  "errors" :
  [
    {
      "schemaPath" : "$.id",
      "instancePath" : "$",
      "code" : "JZN-00503",
      "error" : "invalid type
found, actual: string,
expected: number"
    }
  ]
}
```

JSON in Oracle – Multiple avenues

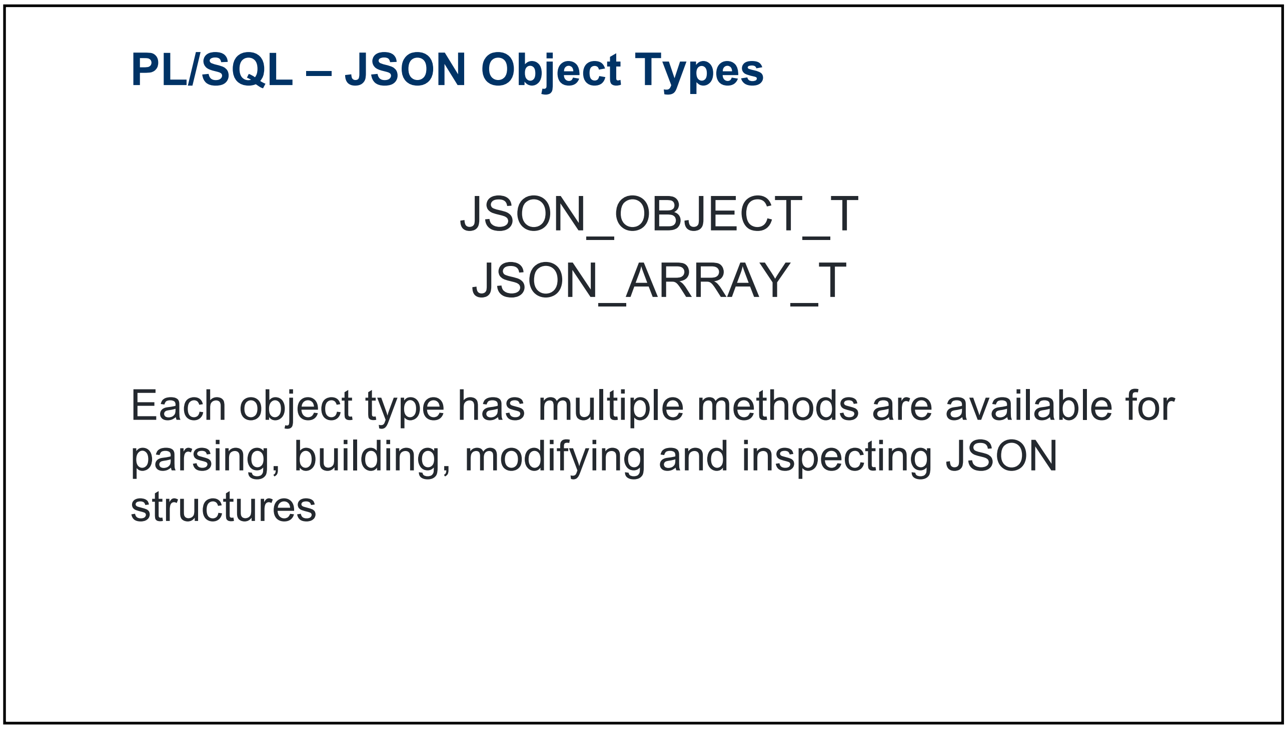
- Oracle SODA – accepts JSON from multiple environments
  - *Java, Node.js, REST, C, Python, PL/SQL*
- Oracle's new MongoDB Drivers and Tools
- REST and ORDS
- SQL and PL/SQL



PL/SQL - APEX_JSON

There is a DB package called APEX_JSON that can be used for parsing and generating JSON – available since APEX 5.0

```
apex_json.open_object();
 apex_json.write('latestStay','2022-10-06’);
 apex_json.openArray('reviews’);
  apex_json.open_object();
   apex_json.write('name','RoOUG Reviewer');
   apex_json.write('stars','5');
   apex_json.write(‘text’,’Lovely place’);
  apex_json.close_object();
 apex_json.close_array();
apex_json.close_object();
```



PL/SQL – JSON Object Types

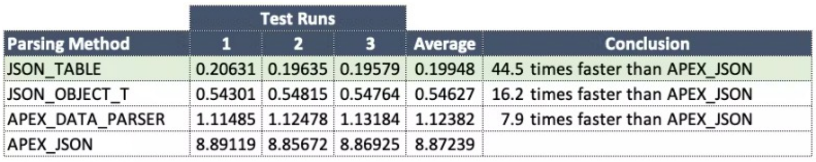
JSON_OBJECT_T
JSON_ARRAY_T

Each object type has multiple methods are available for parsing, building, modifying and inspecting JSON structures

PL/SQL – JSON Object Types vs APEX_JSON

A recent article by Jon Dixon would indicate that the PL/SQL JSON Object types are many times faster than APEX_JSON for parsing JSON.

For Speeds Sake, Stop Using APEX_JSON (Jon Dixon, 5th June 2022)



	Test Runs				
Parsing Method	1	2	3	Average	Conclusion
JSON_TABLE	0.20631	0.19635	0.19579	0.19948	44.5 times faster than APEX_JSON
JSON_OBJECT_T	0.54301	0.54815	0.54764	0.54627	16.2 times faster than APEX_JSON
APEX_DATA_PARSER	1.11485	1.12478	1.13184	1.12382	7.9 times faster than APEX_JSON
APEX_JSON	8.89119	8.85672	8.86925	8.87239	

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

Challenges of Hybrid models

- With less tables and more flexibility - more 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

NoSQL / JSON in the Database

- JSON in the Database is here to stay
- Let’s embrace it and add it to our toolkit

Any Questions?