JSON IN ORACLE

Neil Chandler

Chandler Systems

SPOUG SPAIN ORACLE USERS GROUP

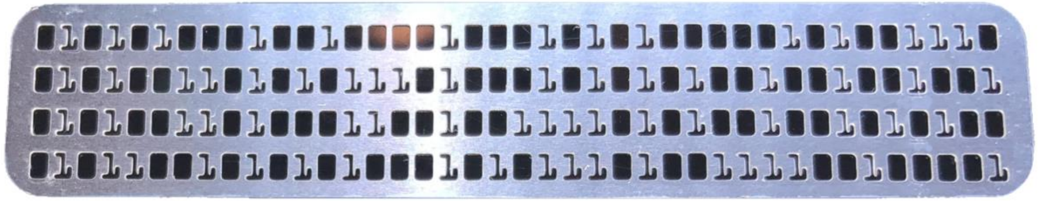
KNOWLEDGE | NETWORKING | INFLUENCE

Technical SPOUG Day

Technical event in the largest Wind Tunnel in Europe, with knowledge from some of the world's best Oracle ACEs, ACE Directors and Rock Star Speakers

25 Sept 2018

JSON IN ORACLE



Neil Chandler
Chandler Systems

Independent Database Consultant
Working in IT for 30 years

ORACLE
ACE Director

BLOG: http://chandlerdba.com
Twit: @chandlerDBA
E: neil@chandler.uk.com

WHAT IS JSON

JSON – JavaScript Object Notation

It is an open-standard file format, based originally on JavaScript object literal notation, that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types

It's an alternative to XML, with less overhead

It's easy to read and easy to parse

WHAT IS JSON

```
{
"firstName":"Elisabeth",
"lastName":"Windsor",
"dob":"1926-04-21",
"age":91,
"alive":true,
"address":
 { "streetAddress": "Buckingham Palace",
   "city": "London",
   "state": "Middlesex",
   "postalCode": "SW1A 1AA" },
"phoneNumber":
 [ { "type": "home",    "number": "+44 (0)20 5555 1234" },
   { "type": "mobile", "number": "+44 (0)7802 555 123" }
 ],
"gender": { "type": "female" },
"preferredBeverage":null,
"children":
[{"name":"Anne"},{"name":"Charles"},{"name":"Andrew"},{"name":"Edward"}]
}
```

Name/Attribute-Value Pair Object

There's no JSON type for a date - use ISO8601 format

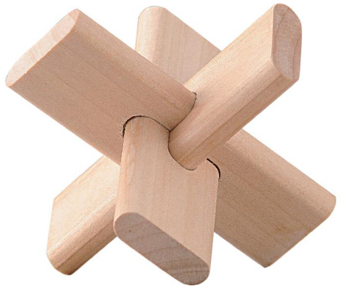
The "value" can be a **string**, **number**, **boolean**, **null**, **object** or array

Square Brackets [ ] signifying an array

WHY JSON?

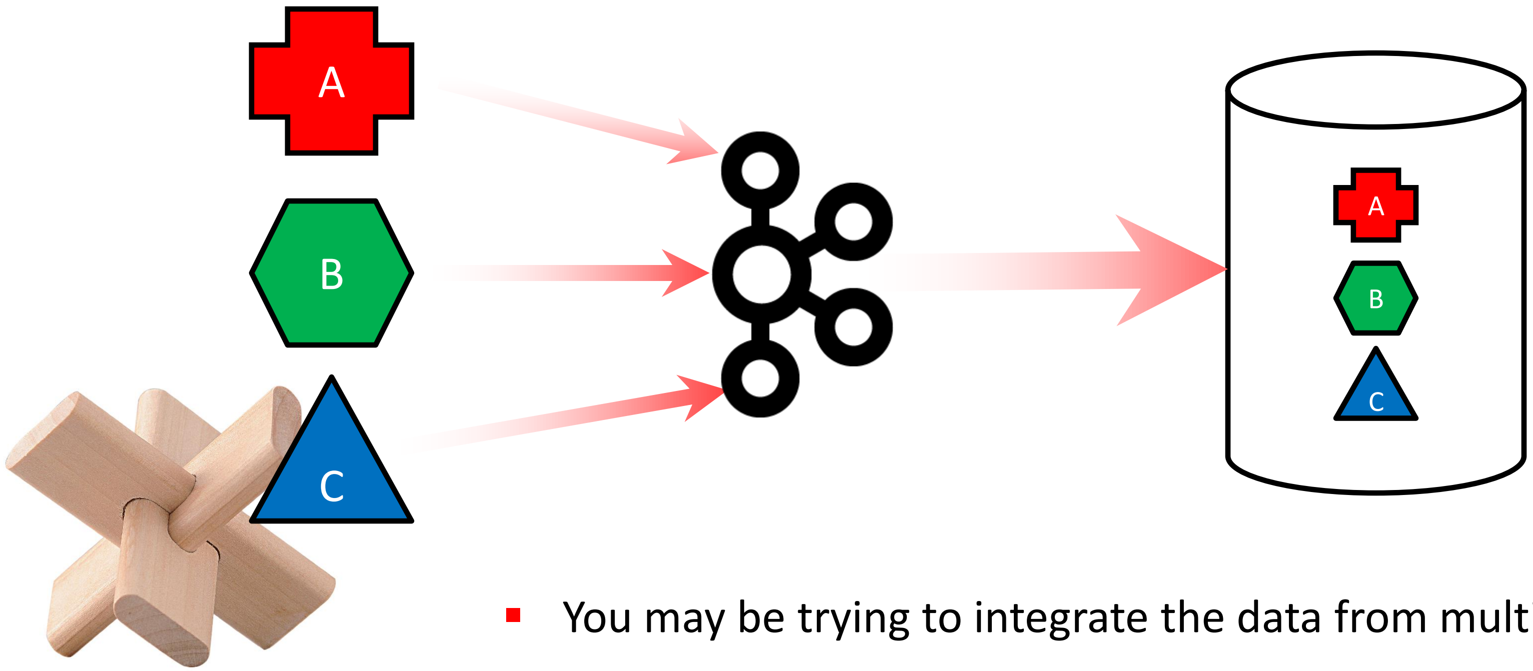
- Your data may already be in JSON format
  - Maybe coming from an IoT device
  - Maybe some micro-service communication
  - If you are using RESTful services, you're probably already using JSON

- Might be hard to map the information to Relational Structure



- You may want (some) schema flexibility
  - Easy to change or enhance a schema
  - Schema may not be known
  - Schema may be from one or more (3rd party) applications
  - You may be trying to integrate the data from multiple applications

WHY JSON



- You may be trying to integrate the data from multiple applications

WHO SUPPORTS JSON

Which databases support JSON? Pretty much all of them..

Dedicated NoSQL Databases

- DynamoDB, Cassandra
  (rigid structure, poor JSON filtering/indexing)
- MongoDB, **Couchbase**, CosmoDB, etc…
  (tends to lack guaranteed **Durability** in favour of horizontal scaling and quick ingestion - memory commits with lazy writes)

Relational Databases which conform to SQL 2016 SQL/JSON standards

- **Oracle 12C**, PostgreSQL, MySQL, MariaDB, SQL Server, DB2

Beware of vendor-specific extensions to JSON processing!

ORACLE AND JSON

Storing JSON

There's no such thing as a JSON data type - it's a constraint

```
create table json_tab
(
 json_data      varchar2(4000)
 CONSTRAINT     json_data_ck CHECK (json_data IS JSON)
);
```

STORAGE DATATYPES

```
create table json_tab
(
 json_data  varchar2(4000)
 CONSTRAINT json_data_ck CHECK (json_data IS JSON)
);
```

Datatypes used to store JSON are **VARCHAR2**, **CLOB** and **BLOB**

- **VARCHAR2 -** limited to 4000 bytes
  [a VARCHAR2(4001-32768) is a CLOB in disguise, but advantages in PL/SQL & InMemory]
- **CLOB** - slower than a varchar2. Stored using UCS2 (like UTF16) - 2 bytes per character
- **BLOB** - uses UTF8's but it's stored in Binary
  A bit less friendly but *may* offer a space saving and performance improvement over CLOB

"When possible, Oracle recommends that you use BLOB storage."

STORAGE DATATYPES

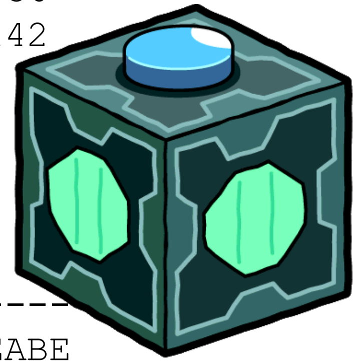
```
SQL> desc json_tab_clob
Name             Null?  Type
---------------- ------ -------
JSON_DATA               CLOB

SQL> select json_data from json_tab_clob where rownum=1;
JSON_DATA
---------------------------------------------------------------------------
{"EMPLOYEE_ID":174,"FIRST_NAME":"Ellen","LAST_NAME":"Abel","EMAIL":"EABE
```

STORAGE DATATYPES

```
SQL> desc json_tab_blob
Name                Null?    Type
------------------- -------- -------
JSON_DATA                    BLOB
```

```
SQL> select * from USER_JSON_COLUMNS;

TABLE_NAME                COLUMN_NAME       FORMAT     DATA_TYPE
------------------------- ----------------- ---------- -------------
JSON_TAB                  JSON_DATA         TEXT       VARCHAR2
JSON_TAB_CLOB             JSON_DATA         TEXT       CLOB
JSON_TAB_BLOB             JSON_DATA         TEXT       BLOB
```

```
SQL> select json_data from json_tab_blob where rownum=1;
JSON_DATA
------------------------------------------------------------------------
7B22454D504C4F5945455F4944223A3137342C224649525354F4E414D45223A22456C6
C656E222C224C4153545F4E414D45223A224162656C222C22454D41494C223A22454142

SQL> select utl_raw.cast_to_varchar2(json_data) from json_tab_blob
     where rownum=1;

UTL_RAW.CAST_TO_VARCHAR2(JSON_DATA)
------------------------------------------------------------------------
{"EMPLOYEE_ID":174,"FIRST_NAME":"Ellen","LAST_NAME":"Abel","EMAIL":"EABE
```

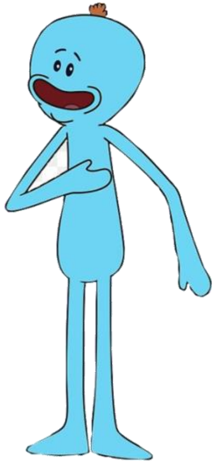


But that's not how we really should select JSON from a JSON field…

DOT NOTATION

```
"firstName":"Elisabeth",
"lastName":"Windsor",
"dob":"1926-04-21",
"gender": { "type": "female" },
"children":[{"name":"Anne"},{"name":"Charles"},{"name":"Andrew"},{"name":"Edward"}]
```

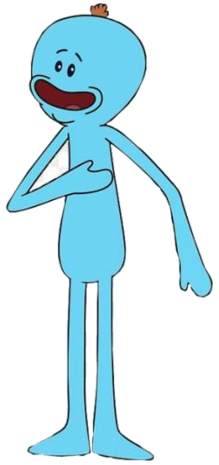
```
SQL> select
       from json_tab jt


FIRSTNAME               COUNT(*)
---------------------- ----------
nulls                        134
Elisabeth                      1
```



DOT NOTATION

```
"firstName":"Elisabeth",
"lastName":"Windsor",
"dob":"1926-04-21",
"gender": { "type": "female" },
"children":[{"name":"Anne"},{"name":"Charles"},{"name":"Andrew"},{"name":"Edward"}]
```

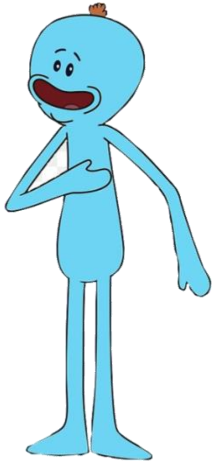
```
SQL> select jt.json_data.firstName,count(*)
       from json_tab jt
      where JSON_EXISTS(jt.json_data,'$[0].firstName')
      group by jt.json_data.firstName;

FIRSTNAME                        COUNT(*)
-------------------- ----------
Elisabeth                               1
```



DOT NOTATION

```
"firstName":"Elisabeth",
"lastName":"Windsor",
"dob":"1926-04-21",
"gender": { "type": "female" },
"children":[{"name":"Anne"},{"name":"Charles"},{"name":"Andrew"},{"name":"Edward"}]
```

```
SQL> select jt.json_data.firstname,count(*)
       from json_tab jt
      group by jt.json_data.firstname;

FIRSTNAME                  COUNT(*)
-------------------- ----------
nulls                           135
```



Dot notation is case sensitive

DOT NOTATION

```
"firstName":"Elisabeth",
"lastName":"Windsor",
"dob":"1926-04-21",
"gender": { "type": "female" },
"children":[{"name":"Anne"},{"name":"Charles"},{"name":"Andrew"},{"name":"Edward"}]
```

```
SQL> select jt.json_data.firstname,count(*)
       from json_tab jt
      group by jt.json_data.FIRSTName;
```

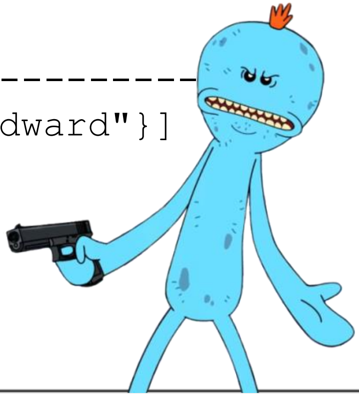
```
select jt.json_data.firstname,count(*)
  from json_tab jt
 group by jt.json_data.FIRSTName

ERROR at line 1:
ORA-00979: not a GROUP BY expression
```

DOT NOTATION

```
"firstName":"Elisabeth",
"lastName":"Windsor",
"dob":"1926-04-21",
"gender": { "type": "female" },
"children":[{"name":"Anne"},{"name":"Charles"},{"name":"Andrew"},{"name":"Edward"}]
```

```
select jt.json_data.children,
       jt.json_data.children.name from json_tab jt
 where JSON_EXISTS(jt.json_data,'$[0].lastName');
```

```
CHILDREN
----------------------------------------------------------------------------
[{"name":"Anne"},{"name":"Charles"},{"name":"Andrew"},{"name":"Edward"}]

CHILDREN
-----------------------------------
["Anne","Charles","Andrew","Edward"]
```



DOT NOTATION

```
"firstName":"Elisabeth",
"lastName":"Windsor",
"dob":"1926-04-21",
"gender": { "type": "female" },
"children":[{"name":"Anne"},{"name":"Charles"},{"name":"Andrew"},{"name":"Edward"}]
```

```
select jt.json_data.gender,
       jt.json_data.gender.type
  from json_tab jt
 where JSON_EXISTS(jt.json_data,'$[0].lastName')
   and jt.json_data.gender.type='female';
```

```
GENDER               GENDER
-------------------- --------------------
{"type":"female"}    female
```

DOT NOTATION

You must use an **IS JSON** or **IS JSON STRICT** constraint

```
SQL> select jt.json_data.firstName from json_tab jt;
select jt.json_data.firstName from json_tab jt
       *
ERROR at line 1:
ORA-00904: "JT"."JSON_DATA"."FIRSTNAME": invalid identifier

SQL> alter table json_tab add constraint json_data_ck check (json_data is JSON);
Table altered.

SQL> select jt.json_data.firstName from json_tab jt;
FIRSTNAME
--------------------
Elisabeth
```

EXTRACTING JSON

But that's all relational. I want my data out as JSON!
We can use the function JSON_OBJECT

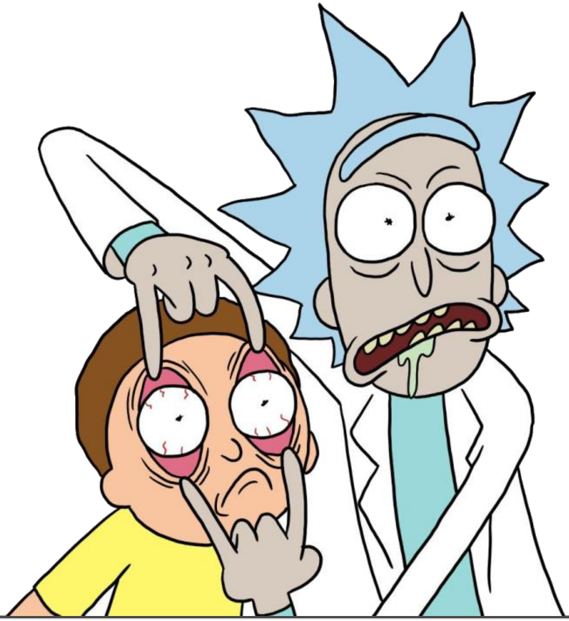
```
select JSON_OBJECT('firstName'         VALUE jt.json_data.FIRST_NAME,
                   'familyName'        VALUE jt.json_data.LAST_NAME,
                   'salary'            VALUE TO_NUMBER(jt.json_data.SALARY) ,
                   'commissionPercent' VALUE jt.json_data.COMMISSION_PCT,
                   'Employee_ID'       VALUE id,
                   'createdDate'       VALUE created_date
                   ) as JSON_EXTRACT
       from json_tab jt where jt.json_data.FIRST_NAME='Vance'

JSON_EXTRACT
----------------------------------------------------------
{"firstName":"Vance","familyName":"Jones","salary":2800,
 "commissionPercent":null,"Employee_ID":457,
 "createdDate":"2018-03-15T16:50:55"}
```

YOUR JSON SCHEMA

- The **Big Problem** with Relational is you should know your schema before you code and shouldn't change it much.
  This is a Barrier to commencement!
  This is a barrier to AGILE!
  (apparently)

- Some Developers believe that JSON is
  schema-never or schema-later or schema-on-read.
  This may be a huge mistake.

- Evolving schemas are great to store simple application data
  BUT they can be difficult/impossible to search
  Understanding your *critical* search requirements
  beforehand is important.



DESIGNING YOUR SCHEMA

```
create table GENERIC_JSON_TABLE
(
 id              NUMBER default json_seq.nextval NOT NULL,
 created_date    DATE   default sysdate not null,
 modified_date   DATE   default sysdate not null,
 version         NUMBER, - are you going to record the doc version
 json_id         NUMBER, - denormalised PK from the document?
                           other attributes you want to denormalise
 JSON_DATA       BLOB
 CONSTRAINT      generic_json_data_ck CHECK (json_data IS JSON)
)
```

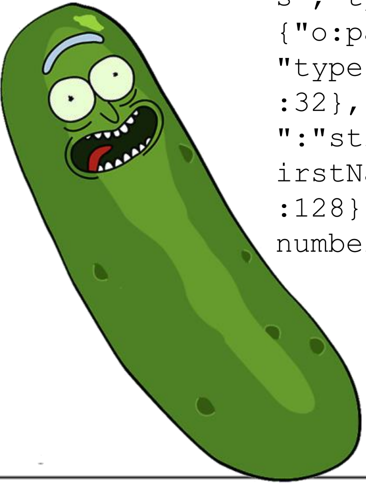
Are you only going to store 1 document type in that table?

JSON_DATAGUIDE

- JSON is agile: Developers add to it, you ingest new systems, it evolves

```
select JSON_DATAGUIDE(json_data) from json_tab;
```

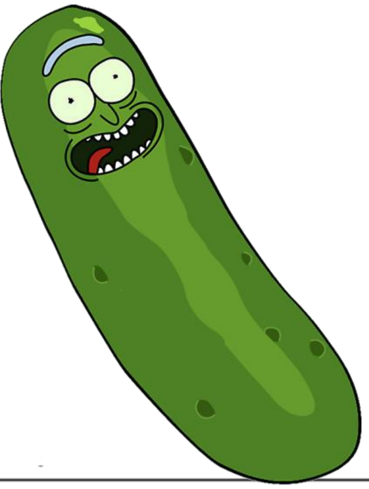
```
[{"o:path":"$.age","type":"number","o:length":2},{"o:path":"$.dob","type":"string","o:length
":16},{"o:path":"$.alive","type":"boolean","o:length":4},{"o:path":"$.gender","type":"object
","o:length":32},{"o:path":"$.gender.type","type":"string","o:length":8},{"o:path":"$.addres
s","type":"object","o:length":128},{"o:path":"$.address.city","type":"string","o:length":8},
{"o:path":"$.address.state","type":"string","o:length":16},{"o:path":"$.address.postalCode",
"type":"string","o:length":8},{"o:path":"$.address.streetAddress","type":"string","o:length"
:32},{"o:path":"$.children","type":"array","o:length":128},{"o:path":"$.children.name","type
":"string","o:length":8},{"o:path":"$.lastName","type":"string","o:length":8},{"o:path":"$.f
irstName","type":"string","o:length":16},{"o:path":"$.phoneNumber","type":"array","o:length"
:128},{"o:path":"$.phoneNumber.type","type":"string","o:length":8},{"o:path":"$.phoneNumber.
number","type":"string","o:length":32}]
```



JSON_DATAGUIDE

- JSON is agile: Developers add to it, you ingest new systems, it evolves

```
select JSON_DATAGUIDE(json_data) from json_tab;
```



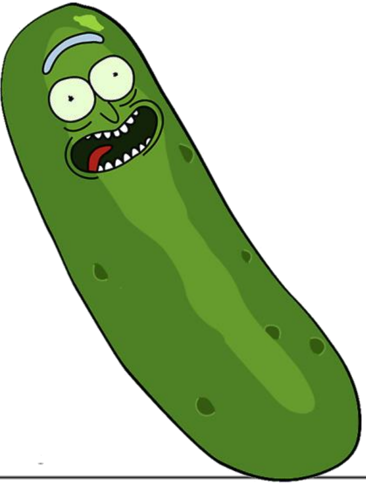
```
[{
        "o:path": "$.age",
        "type": "number",
        "o:length": 2
}, {
        "o:path": "$.dob",
        "type": "string",
        "o:length": 16
}, {
        "o:path": "$.alive",
        "type": "boolean",
        "o:length": 4
}, {
        "o:path": "$.gender",
        "type": "object",
        "o:length": 32
}, {
        "o:path": "$.gender.type",
```

JSON_DATAGUIDE

- JSON is agile: Developers add to it, you ingest new systems, it evolves

```
select JSON_DATAGUIDE(json_data) from json_tab;
```

You may get duplicates if the path has multiple incompatible types

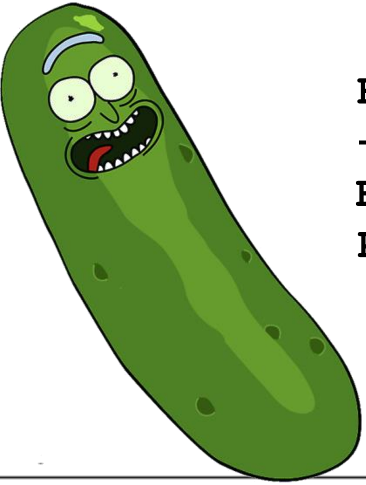


```
{
 "o:path" : "$.age",
 "type" : "number",
 "o:length" : 2,
},
{
 "o:path" : "$.age",
 "type" : "object",
 "o:length" : 32,
 },
{
  "o:path" : "$.age.old",
  "type" : "string",
  "o:length" : 16,
},
```

JSON_DATAGUIDE

- JSON is agile: Developers add to it, you ingest new systems, it evolves

```
select jt.json_data.firstName,jt.json_data.lastName,
       jt.json_data.age from json_tab jt
 where JSON_EXISTS(jt.json_data,'$[0].age')
```



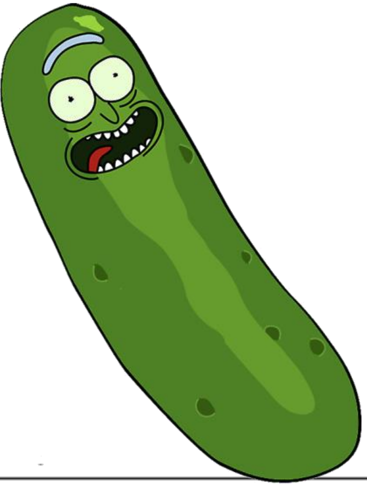
```
FIRSTNAME  LASTNAME  AGE
---------- --------- --------------------
Elizabeth  Windsor   91
Phillip    Windsor   {"old":"ninety six"}
```

JSON_DATAGUIDE

- JSON is agile: Developers add to it, you ingest new systems, it evolves

```
select JSON_DATAGUIDE(json_data) from json_tab;
```

```
When you re-run this, you may get new attributes appearing
```

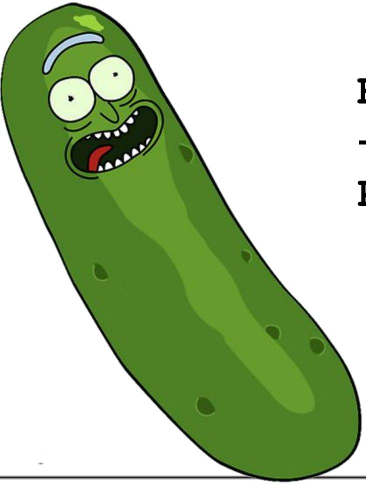


```
                    "type": "string","o:length": 4
        },{
                    "o:path": "$.hobby",
                    "type": "string",
                    "o:length": 16
        },{
                    "o:path": "$.gender",
                    "type": "object",
                    "o:length": 32
        },{
                    "o:path": "$.gender.type",
                    "type": "string",
                    "o:length": 8
        },{
```

JSON_DATAGUIDE

- JSON is agile: Developers add to it, you ingest new systems,ito evolves

```
select jt.json_data.firstName,jt.json_data.lastName,
       jt.json_data.hobby from json_tab jt
 where JSON_EXISTS(jt.json_data,'$[0].hobby')
```



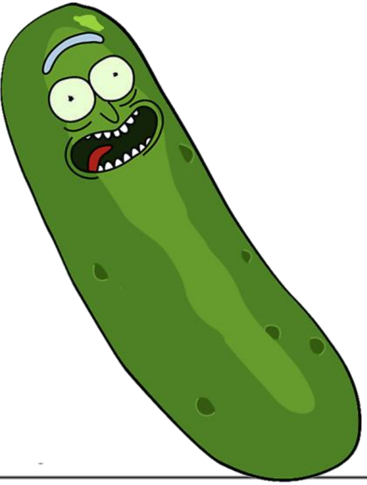
```
FIRSTNAME  LASTNAME  HOBBY
---------- --------- -----------------
Phillip    Windsor   Hunting Peasants
```

JSON_DATAGUIDE

Maintaining the Dataguide:

```
create search index JSON_TAB_SIDX
       on JSON_TAB (JSON_DATA)
     for JSON PARAMETERS ('SEARCH_ON NONE DATAGUIDE ON');

select * from USER_JSON_DATAGUIDES; [in 18C: USER_JSON_DATAGUIDE_FIELDS for relational format]
```

WARNING!
Maintained on "commit"

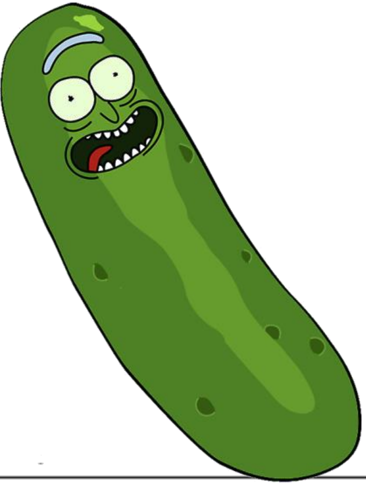


```
TABLE_NAME COLUMN_NAME DATAGUIDE
---------- ----------- ----------------------------------------------
JSON_TAB   JSON_DATA   [
                         {
                          "o:path" : "$.age",
                          "type" : "number",
                          "o:length" : 2,
                          "o:preferred_column_name" : "JSON_DATA$age"
                         },
                         {
                          "o:path" : "$.dob",
                          "type" : "string",
                          "o:length" : 16,
                          "o:preferred_column_name" : "JSON_DATA$dob"
```

JSON_DATAGUIDE

Maintaining the Dataguide:

```
select dbms_json.get_index_dataguide
       ('neil','json_tab','json_data',dbms_json.format_hierarchical) guide from dual;
```



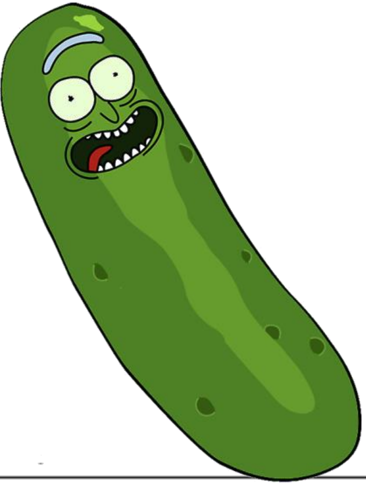
We can use this dataguide to do some cool stuff…

JSON_DATAGUIDE - VIEW

Creating a View from the dataguide

```
exec dbms_json.create_view_on_path('JSON_VIEW','JSON_TAB','JSON_DATA', '$');
```

```
SQL > desc json_view
 Name                              Null?    Type
 --------------------------------- -------- --------------------
 ID                                NOT NULL NUMBER
 CREATED_DATE                      NOT NULL DATE
 JSON_DATA$age                              NUMBER
 JSON_DATA$old                              VARCHAR2(16)
 JSON_DATA$dob                              VARCHAR2(16)
 JSON_DATA$alive                            VARCHAR2(4)
 JSON_DATA$hobby                            VARCHAR2(16)
 JSON_DATA$type                             VARCHAR2(8)
 JSON_DATA$city                             VARCHAR2(8)
 JSON_DATA$state                            VARCHAR2(16)
 JSON_DATA$postalCode                       VARCHAR2(8)
 JSON_DATA$streetAddress                    VARCHAR2(32)
 JSON_DATA$lastName                         VARCHAR2(8)
 JSON_DATA$firstName                        VARCHAR2(16)
 JSON_DATA$name                             VARCHAR2(8)
 JSON_DATA$type_2                           VARCHAR2(8)
 JSON_DATA$number                           VARCHAR2(32)
```



JSON_DATAGUIDE - VIEW

Creating a View from the dataguide

```
SQL > select count(*) from json_tab;

  COUNT(*)
----------
         2

SQL > select count(*) from json_view;

  COUNT(*)
----------
        13
```

JSON_DATAGUIDE - VIEW

Creating a View from the dataguide

```
select id,"JSON_DATA$firstName","JSON_DATA$lastName", "JSON_DATA$dob",  "JSON_DATA$hobby",
          "JSON_DATA$name",      "JSON_DATA$type_2",   "JSON_DATA$number"
  from json_view;

 ID JSON_DATA$ JSON_DAT JSON_DATA$ JSON_DATA$hobby JSON_DATA$name JSON_DATA$type_2 JSON_DATA$number
--- ---------- -------- ---------- ---------------- ------------- ---------------- -------------------
  1 Elisabeth  Windsor  1926-04-21                  Andrew
  1 Elisabeth  Windsor  1926-04-21                  Anne
  1 Elisabeth  Windsor  1926-04-21                  Charles
  1 Elisabeth  Windsor  1926-04-21                  Edward
  1 Elisabeth  Windsor  1926-04-21                                home             +44 (0)20 5555 1234
  1 Elisabeth  Windsor  1926-04-21                                mobile           +44 (0)7802 555 123
  4 Phillip    Windsor  1921-06-21 hunting peasants               home             +44 (0)20 5555 1234
  4 Phillip    Windsor  1921-06-21                  Andrew
  4 Phillip    Windsor  1921-06-21                  Anne
  4 Phillip    Windsor  1921-06-21                  Charles
  4 Phillip    Windsor  1921-06-21                  Edward
  4 Phillip    Windsor  1921-06-21                                fax              +44 (0)20 5555 4321
  4 Phillip    Windsor  1921-06-21                                mobile           +44 (0)7802 555 123
```

JSON_DATAGUIDE - VIEW

```
SELECT
    "RT"."ID",                        "RT"."CREATED_DATE",
    jt."JSON_DATA$age",                jt."JSON_DATA$old",        jt."JSON_DATA$dob",
    jt."JSON_DATA$alive",              jt."JSON_DATA$hobby",      jt."JSON_DATA$type",
    jt."JSON_DATA$city",               jt."JSON_DATA$state",      jt."JSON_DATA$postalCode",
    jt."JSON_DATA$streetAddress",      jt."JSON_DATA$lastName",   jt."JSON_DATA$firstName",
    jt."JSON_DATA$name",               jt."JSON_DATA$type_2",     jt."JSON_DATA$number"
FROM
    JSON_TAB rt,
    JSON_TABLE ( "JSON_DATA" FORMAT JSON,'$'
            COLUMNS
                "JSON_DATA$age" NUMBER PATH '$.age',
                "JSON_DATA$old" VARCHAR2 ( 16 ) PATH '$.age.old',
                snip
                "JSON_DATA$streetAddress" VARCHAR2 ( 32 ) PATH '$.address.streetAddress',
                NESTED PATH '$.children[*]'
                    COLUMNS (
                        "JSON_DATA$name" VARCHAR2 ( 8 ) PATH '$.name'
                    ),
                "JSON_DATA$lastName" VARCHAR2 ( 8 ) PATH '$.lastName',
                "JSON_DATA$firstName" VARCHAR2 ( 16 ) PATH '$.firstName',
                NESTED PATH '$.phoneNumber[*]'
                    COLUMNS (
                        "JSON_DATA$type_2" VARCHAR2 ( 8 ) PATH '$.type',
                        "JSON_DATA$number" VARCHAR2 ( 32 ) PATH '$.number'
                    )) jt
```

JSON_DATAGUIDE - VC

Creating Virtual Columns from the dataguide

```
SQL > exec dbms_json.add_virtual_columns('json_tab','json_data',20);
PL/SQL procedure successfully completed.

SQL > desc json_tab
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ID                                        NOT NULL NUMBER
 CREATED_DATE                              NOT NULL DATE
 JSON_DATA                                          VARCHAR2(4000)
 JSON_DATA$age                                      NUMBER
 JSON_DATA$cid                                      VARCHAR2(16)
 JSON_DATA$dob                                      VARCHAR2(16)
 JSON_DATA$alive                                    VARCHAR2(4)
 JSON_DATA$hobby                                    VARCHAR2(16)
 JSON_DATA$type                                     VARCHAR2(8)
 JSON_DATA$city                                     VARCHAR2(8)
 JSON_DATA$state                                    VARCHAR2(16)
 JSON_DATA$postalCode                               VARCHAR2(8)
 JSON_DATA$streetAddress                            VARCHAR2(32)
 JSON_DATA$lastName                                 VARCHAR2(8)
 JSON_DATA$firstName                                VARCHAR2(16)
```

NOTE: scalar (flat) columns only!

add_virtual_columns can have a "frequency" as a 3rd input:
Attributes need to appear in frequency% of documents to be included

JSON_DATAGUIDE - VC

The virtual columns can be accessed just like relational columns

```
SQL > select id,"JSON_DATA$firstName", "JSON_DATA$lastName",
                "JSON_DATA$age",        "JSON_DATA$old",
                "JSON_DATA$hobby"
        from json_tab
       where "JSON_DATA$lastName"='Windsor';


 ID JSON_DATA$ JSON_DAT JSON_DATA$age JSON_DATA$old    JSON_DATA$hobby
--- ---------- -------- ------------- ---------------- ----------------
  1 Elisabeth  Windsor             91
  4 Phillip    Windsor                ninety six       hunting peasants


------------------------------------------------------------------------------
| Id  | Operation         | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |          |     2 |   124 |     3   (0)| 00:00:01 |
|*  1 |  TABLE ACCESS FULL| JSON_TAB |     2 |   124 |     3   (0)| 00:00:01 |
------------------------------------------------------------------------------

   1 - filter("JSON_DATA$lastName"='Windsor')
```

JSON_DATAGUIDE - CHANGE

Creating Virtual Columns from the dataguide

```
SQL > create search index JSON_TAB_SIDX
          on JSON_TAB (JSON_DATA)
          for JSON PARAMETERS
         ('SEARCH_ON NONE
          DATAGUIDE ON CHANGE add_vc');
```

```
SQL > insert into json_tab(json_data)
      values ('{"NewAttribute":"Hello World"}');
```

```
SQL > desc json_tab
 Name                          Type
 ------------------------- --------------
 ID                            NUMBER
 CREATED_DATE                  DATE
 JSON_DATA                     VARCHAR2(4000)
 JSON_DATA$age                 NUMBER
 .
 .
 JSON_DATA$lastName            VARCHAR2(8)
 JSON_DATA$firstName           VARCHAR2(16)
```

```
SQL > desc json_tab
 Name                          Type
 ------------------------- --------------
 ID                            NUMBER
 CREATED_DATE                  DATE
 JSON_DATA                     VARCHAR2(4000)
 JSON_DATA$age                 NUMBER
 .
 .
 JSON_DATA$lastName            VARCHAR2(8)
 JSON_DATA$firstName           VARCHAR2(16)
 JSON_DATA$NewAttribute        VARCHAR2(16)
```

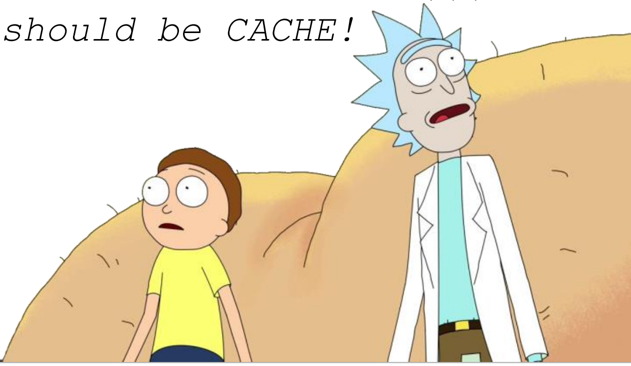
Virtual Columns will not be dynamically removed following a delete

PARTITIONING

- You *can* partition on a JSON attribute by exposing it as a Virtual Column
- You must use the JSON_VALUE with RETURNING clause in the Virtual column to get the datatype right

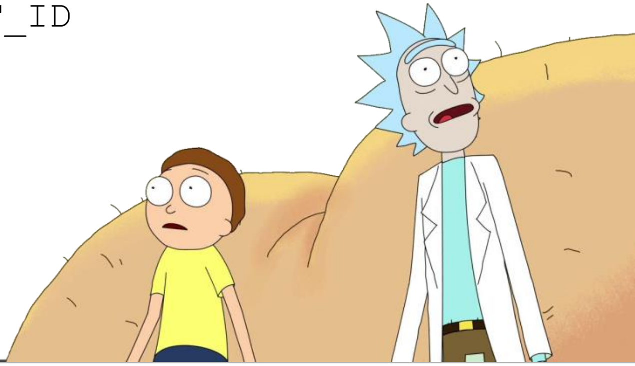
The json_value must be evaluated for every insert.
This can present a scalability problem.

```
create table PART_JSON_TAB
(id              NUMBER default json_seq.nextval NOT NULL PRIMARY KEY,
 created_date    DATE   default sysdate not null,
 PART_JSON_DATA CLOB
 CONSTRAINT      pjt_ck CHECK (part_json_data IS JSON),
 EMP_ID_VC       NUMBER GENERATED ALWAYS AS
                (json_value (PART_JSON_DATA, '$.EMPLOYEE_ID' RETURNING NUMBER)))
 LOB            (PART_JSON_DATA) STORE AS (CACHE)  -- LOB should be CACHE!
 PARTITION BY RANGE (emp_id_vc)
  (PARTITION p1 VALUES LESS THAN (100),
   PARTITION p2 VALUES LESS THAN (200),
   PARTITION p3 VALUES LESS THAN (300));
```



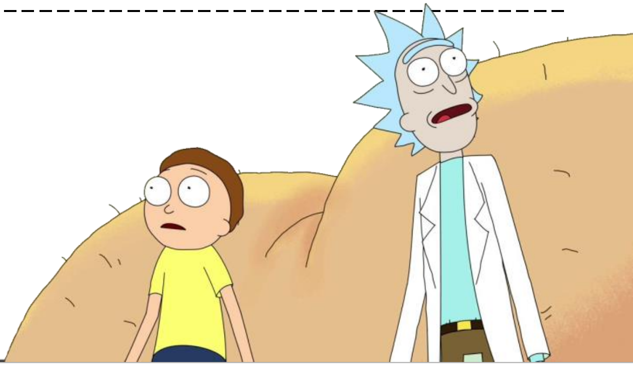
PARTITIONING

```
insert into PART_json_tab (part_json_data)
select json_object('EMPLOYEE_ID'    VALUE  EMP.EMPLOYEE_ID ,
                   'FIRST_NAME'     VALUE  EMP.FIRST_NAME  ,
                   'LAST_NAME'      VALUE  EMP.LAST_NAME   ,
                   'EMAIL'          VALUE  EMP.EMAIL ,
                   'PHONE_NUMBER'   VALUE  EMP.PHONE_NUMBER ,
                   'HIRE_DATE'      VALUE  EMP.HIRE_DATE,
                   'JOB_ID'         VALUE  EMP.JOB_ID ,
                   'SALARY'         VALUE  EMP.SALARY ,
                   'COMMISSION_PCT' VALUE  EMP.COMMISSION_PCT ,
                   'MANAGER_ID'     VALUE  EMP.MANAGER_ID ,
                   'DEPARTMENT_ID'  VALUE  EMP.DEPARTMENT_ID
                  ) from hr.employees emp
order by emp.last_name
/

exec dbms_stats.gather_schema_stats('NEIL');
```



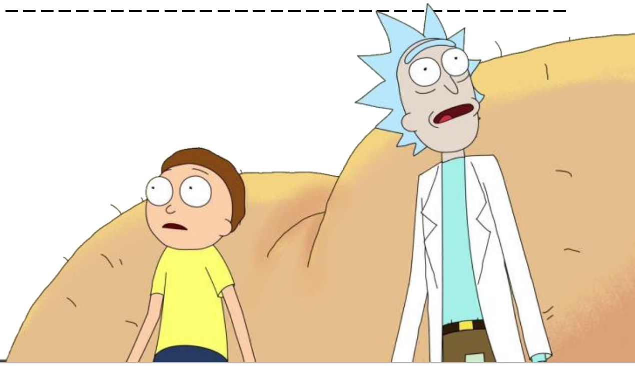
PARTITIONING - VC

```
SQL > select emp_id_vc,part_json_data from part_json_tab pjt
       where emp_id_vc=195;

 EMP_ID_VC PART_JSON_DATA
---------- ----------------------------------------------------------------------------------
       195 {"EMPLOYEE_ID":195,"FIRST_NAME":"Vance","LAST_NAME":"Jones","EMAIL":"VJONES","PH


-------------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name          | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |               |       |       |   276 (100)|          |       |       |
|   1 |  PARTITION RANGE SINGLE|               |     1 |   604 |   276   (0)| 00:00:01 |     2 |     2 |
|*  2 |   TABLE ACCESS FULL    | PART_JSON_TAB |     1 |   604 |   276   (0)| 00:00:01 |     2 |     2 |
-------------------------------------------------------------------------------------------------------------

 2 - filter(JSON_VALUE("PART_JSON_DATA" FORMAT JSON , '$.EMPLOYEE_ID'
           RETURNING NUMBER NULL ON ERROR)=195)
```



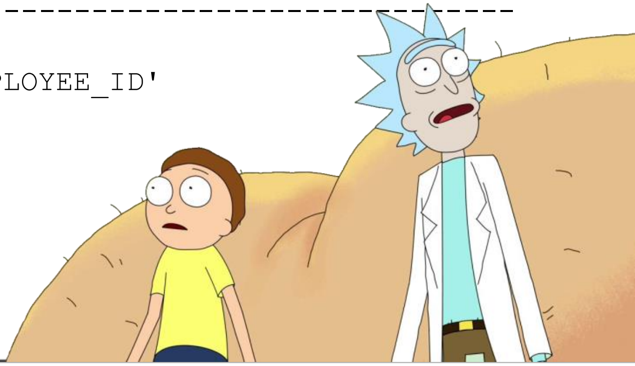
PARTITIONING - JSON_VALUE

```
SQL > select emp_id_vc,part_json_data from part_json_tab pjt
       where json_value(part_json_data, '$.EMPLOYEE_ID' RETURNING NUMBER)=195;

 EMP_ID_VC PART_JSON_DATA
---------- -------------------------------------------------------------------------------
       195 {"EMPLOYEE_ID":195,"FIRST_NAME":"Vance","LAST_NAME":"Jones","EMAIL":"VJONES","PH


------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name           | Rows  | Bytes | Cost  (%CPU)| Time      | Pstart| Pstop |
------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                |       |       |   276 (100)|           |       |       |
|   1 |  PARTITION RANGE SINGLE|                |     1 |   604 |   276   (0)| 00:00:01 |     2 |     2 |
|*  2 |   TABLE ACCESS FULL    | PART_JSON_TAB |     1 |   604 |   276   (0)| 00:00:01 |     2 |     2 |
------------------------------------------------------------------------------------------------------

 2 - filter(JSON_VALUE("PART_JSON_DATA" FORMAT JSON , '$.EMPLOYEE_ID'
              RETURNING NUMBER NULL ON ERROR)=195)
```



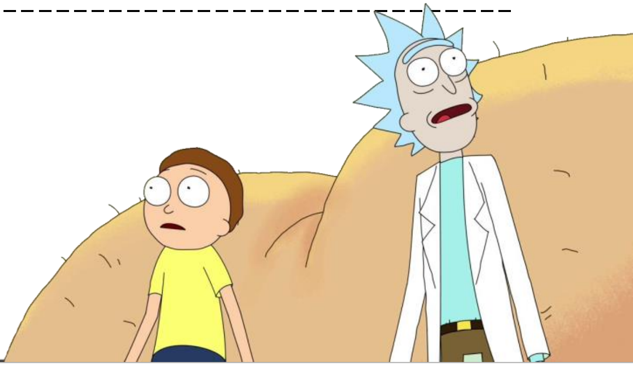
PARTITIONING - JSON_VALUE

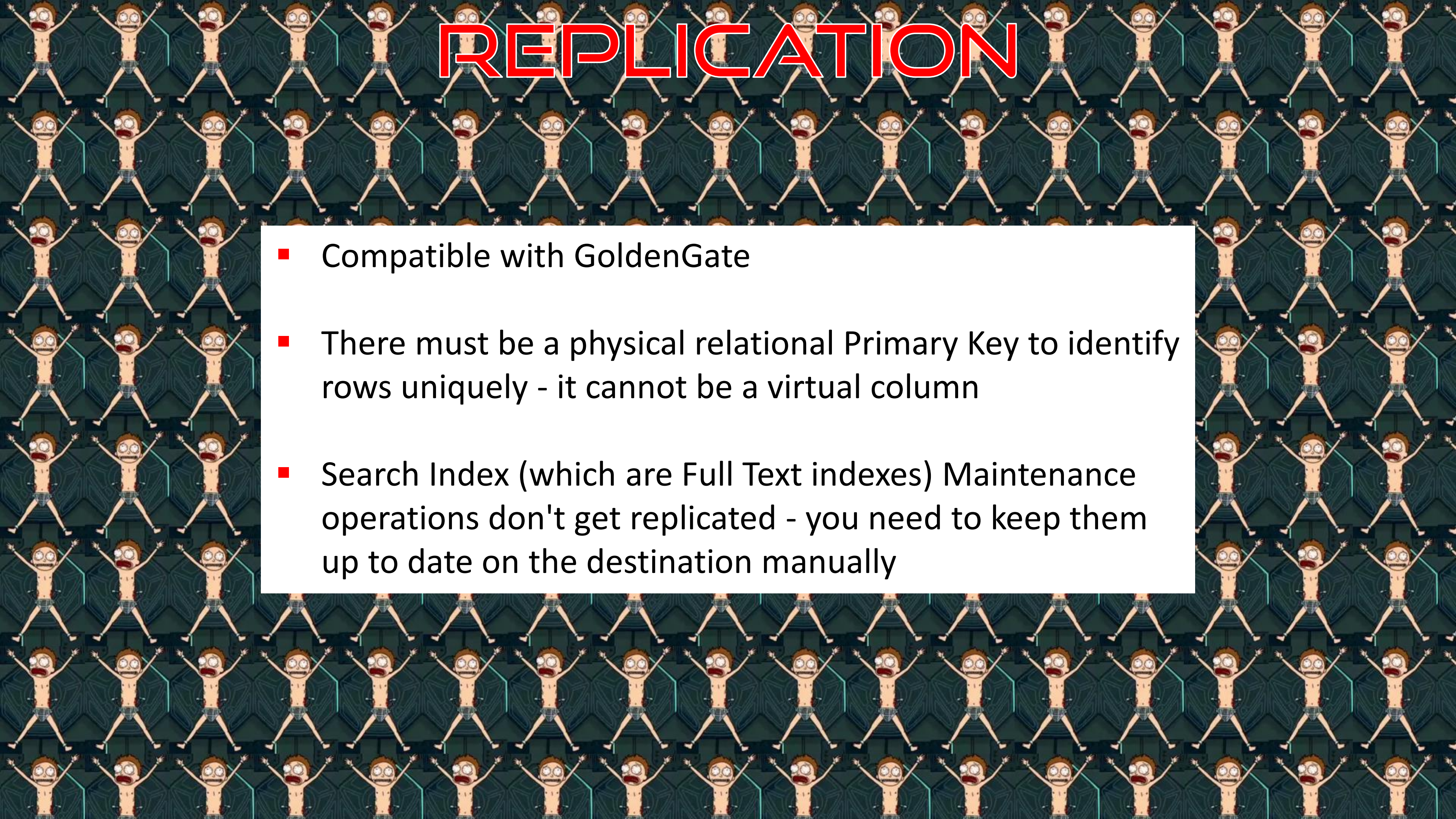
```
SQL > select emp_id_vc,part_json_data from part_json_tab pjt
       where json_value(part_json_data, '$.EMPLOYEE_ID')=195;

 EMP_ID_VC PART_JSON_DATA
---------- --------------------------------------------------------------------------------
       195 {"EMPLOYEE_ID":195,"FIRST_NAME":"Vance","LAST_NAME":"Jones","EMAIL":"VJONES","PH


----------------------------------------------------------------------------------------------------
| Id  | Operation             | Name            | Rows   | Bytes | Cost  (%CPU)| Time      | Pstart| Pstop |
----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                 |        |       |   549 (100)|            |       |       |
|   1 |  PARTITION RANGE ALL|                 |      1 |   604 |   549   (0)| 00:00:01 |     1 |     3 |
|*  2 |   TABLE ACCESS FULL | PART_JSON_TAB |      1 |   604 |   549   (0)| 00:00:01 |     1 |     3 |
----------------------------------------------------------------------------------------------------

   2 - filter(TO_NUMBER(JSON_VALUE("PART_JSON_DATA" FORMAT JSON , '$.EMPLOYEE_ID'
                    RETURNING VARCHAR2(4000) NULL ON ERROR))=195)
```



PARTITIONING - DOT NOTATION

```
SQL > select emp_id_vc,part_json_data from part_json_tab pjt
        where pjt.part_json_data.EMPLOYEE_ID=195;

 EMP_ID_VC PART_JSON_DATA
---------- ---------------------------------------------------------------------
       195 {"EMPLOYEE_ID":195,"FIRST_NAME":"Vance","LAST_NAME":"Jones","EMAIL":"VJONES","PH


-------------------------------------------------------------------------------------------------------
| Id  | Operation           | Name           | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                |       |       |   549 (100)|          |       |       |
|   1 |  PARTITION RANGE ALL|                |     1 |   604 |   549   (0)| 00:00:01 |     1 |     3 |
|*  2 |   TABLE ACCESS FULL | PART_JSON_TAB  |     1 |   604 |   549   (0)| 00:00:01 |     1 |     3 |
-------------------------------------------------------------------------------------------------------

   2 - filter(TO_NUMBER(JSON_QUERY("PJT"."PART_JSON_DATA" FORMAT JSON ,
              '$.EMPLOYEE_ID'RETURNING VARCHAR2(4000)
              ASIS  WITHOUT ARRAY WRAPPER NULL ON ERROR))=195)
```

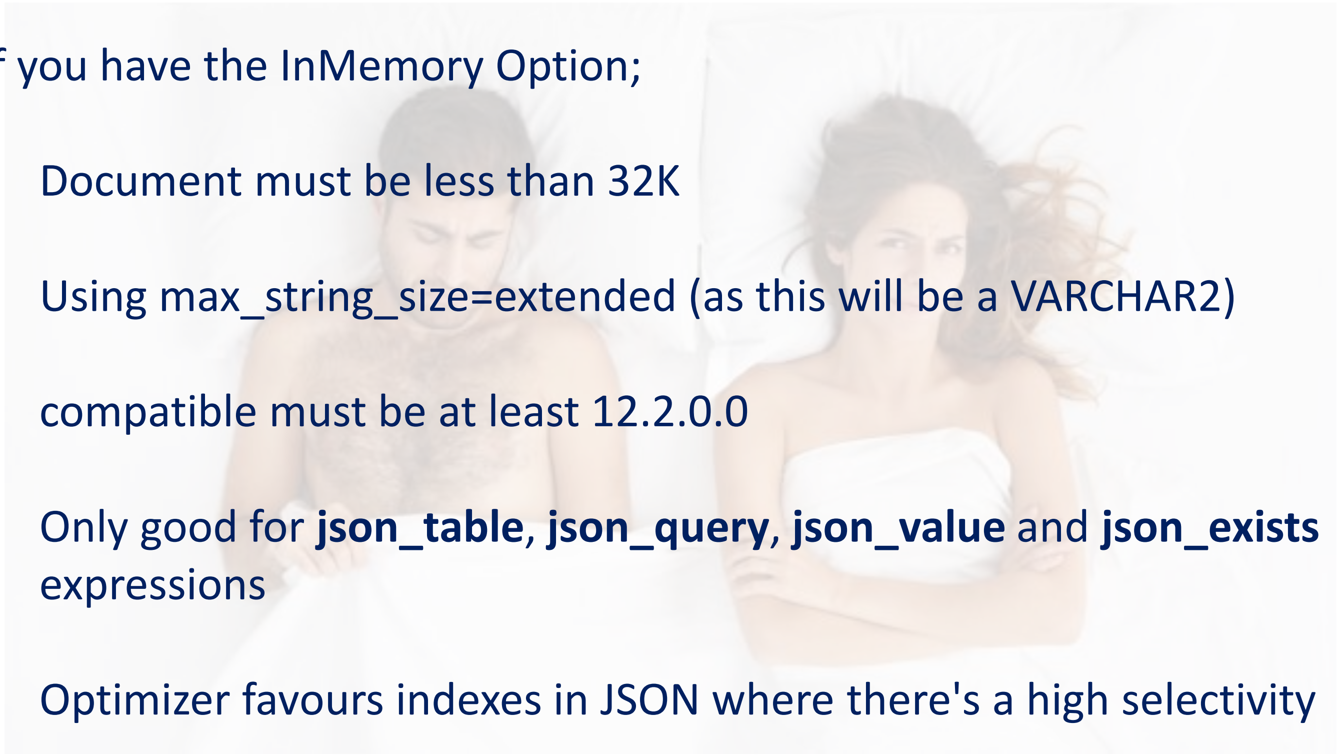




REPLICATION

- Compatible with GoldenGate
- There must be a physical relational Primary Key to identify rows uniquely - it cannot be a virtual column
- Search Index (which are Full Text indexes) Maintenance operations don't get replicated - you need to keep them up to date on the destination manually

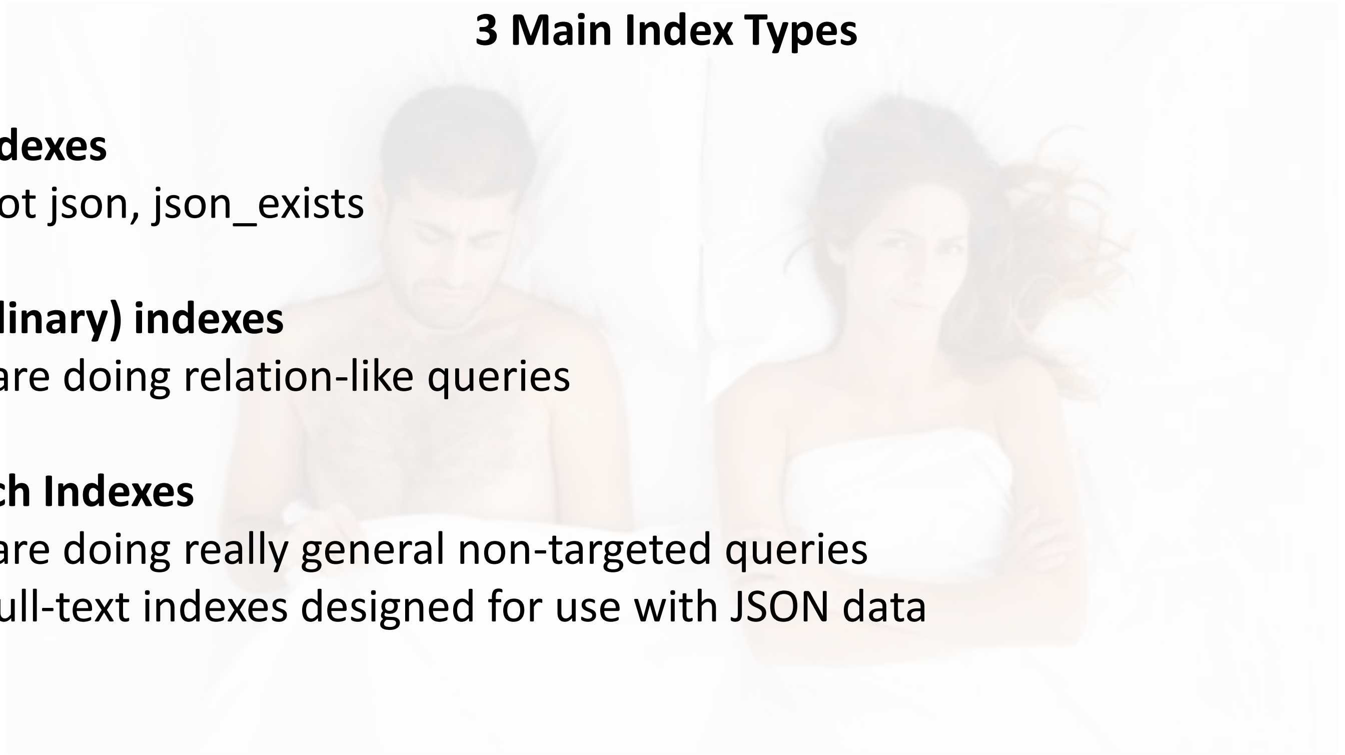
PERFORMANCE



If you have the InMemory Option;

- Document must be less than 32K
- Using max_string_size=extended (as this will be a VARCHAR2)
- compatible must be at least 12.2.0.0
- Only good for **json_table**, **json_query**, **json_value** and **json_exists** expressions
- Optimizer favours indexes in JSON where there's a high selectivity

PERFORMANCE - INDEXING



3 Main Index Types

- **BITMAP indexes**
  is json, is not json, json_exists

- **B-Tree (ordinary) indexes**
  when you are doing relation-like queries

- **JSON Search Indexes**
  when you are doing really general non-targeted queries
  these are full-text indexes designed for use with JSON data

PERFORMANCE - DATA SET

```
SQL > select DATAGUIDE from user_json_dataguides;
```

```
[ {  "o:path" : "$.OWNER",
   "type" : "string",
   "o:length" : 32,
   "o:preferred_column_name" : "JSON_DATA$OWNER"  },
 {  "o:path" : "$.EXISTS",
   "type" : "boolean",
   "o:length" : 8,
   "o:preferred_column_name" : "JSON_DATA$EXISTS"  },
 {  "o:path" : "$.STATUS",
   "type" : "string",
   "o:length" : 8,
   "o:preferred_column_name" : "JSON_DATA$STATUS"  },
 {  "o:path" : "$.CREATED",
   "type" : "string",
   "o:length" : 32,
   "o:preferred_column_name" : "JSON_DATA$CREATED"  },
 {  "o:path" : "$.DELETED",
   "type" : "string",
   "o:length" : 32,
   "o:preferred_column_name" : "JSON_DATA$DELETED"  },
 {  "o:path" : "$.OBJECT_NAME",
   "type" : "string",
   "o:length" : 128,
   "o:preferred_column_name" : "JSON_DATA$OBJECT_NAME"  },
 {  "o:path" : "$.SHARING",
   "type" : "string",
   "o:length" : 32,
   "o:preferred_column_name" : "JSON_DATA$SHARING"  },
 {  "o:path" : "$.GENERATED",
   "type" : "string",
   "o:length" : 1,
   "o:preferred_column_name" : "JSON_DATA$GENERATED"  },
 {  "o:path" : "$.NAMESPACE",
   "type" : "string",
   "o:length" : 4,
   "o:preferred_column_name" : "JSON_DATA$NAMESPACE"  },
 {  "o:path" : "$.OBJECT_ID",
   "type" : "string",
   "o:length" : 8,
   "o:preferred_column_name" : "JSON_DATA$OBJECT_ID"  },
.
.
```

PERFORMANCE - DATA SET

```
SQL> select JSON_DATA$OWNER,  JSON_DATA$OBJECT_NAME,JSON_DATA$OBJECT_TYPE,
            JSON_DATA$CREATED,JSON_DATA$DELETED,    JSON_DATA$EXISTS  from json_view
      where JSON_DATA$OWNER = 'HR' order by 1 ,2;

JSON_DATA$OWNER JSON_DATA$OBJECT_NAME JSON_DATA$OBJECT_TYPE JSON_DATA$CREATED   JSON_DATA$DELETED
--------------- --------------------- --------------------- -----------------   -----------------
HR              ADD_JOB_HISTORY       PROCEDURE             2017-03-02T08:41:44
HR              COUNTRIES             TABLE                 2017-03-02T08:41:43
HR              COUNTRY_C_ID_PK       INDEX                 2017-03-02T08:41:43
HR              DEPARTMENTS           TABLE                 2017-03-02T08:41:43
HR              DEPARTMENTS_SEQ       SEQUENCE              2017-03-02T08:41:43
.
.
.
```

PERFORMANCE - BITMAP

```
SQL> select jt.json_data.OWNER,jt.json_data.OBJECT_NAME,
            jt.json_data.OBJECT_TYPE, jt.json_data.CREATED,jt.json_data.DELETED
       from json_tab jt where json_exists(json_data,'$[0].DELETED')

OWNER      OBJECT_NAM OBJECT_TYP CREATED              DELETED
---------- ---------- ---------- -------------------- --------------------
CHANDLER   OBJECT1    JSON       2018-01-01T00:00:00  2018-02-02T00:00:00
CHANDLER   OBJECT2    JSON       2018-01-01T00:00:00  2018-02-02T00:00:00

-----------------------------------------------------------------------------------
| Id  | Operation               | Name     | Rows   | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |          |        |       | 1071K(100)|          |
|   1 |  NESTED LOOPS           |          |   321M|   179G| 1071K   (1)| 00:00:42 |
|*  2 |   TABLE ACCESS FULL     | JSON_TAB | 39371 |    22M| 1913    (1)| 00:00:01 |
|   3 |   JSONTABLE EVALUATION |          |        |       |            |          |
-----------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter(JSON_EXISTS2("JSON_DATA" FORMAT JSON , '$[0].DELETED' FALSE ON ERROR)=1)
```

PERFORMANCE - BITMAP

```
SQL> CREATE BITMAP INDEX json_data_deleted_idx ON json_tab (json_exists(json_data,'$[0].DELETED'));

SQL> select jt.json_data.OWNER,jt.json_data.OBJECT_NAME,
            jt.json_data.OBJECT_TYPE, jt.json_data.CREATED,jt.json_data.DELETED
       from json_tab jt where json_exists(json_data,'$[0].DELETED')

OWNER      OBJECT_NAM OBJECT_TYP CREATED              DELETED
---------- ---------- ---------- -------------------- --------------------
CHANDLER   OBJECT1    JSON       2018-01-01T00:00:00  2018-02-02T00:00:00
CHANDLER   OBJECT2    JSON       2018-01-01T00:00:00  2018-02-02T00:00:00

--------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name                  | Rows  | Bytes |Cost (%CPU)|Time     |
--------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                       |       |       |   58 (100)|         |
|   1 |  NESTED LOOPS                        |                       | 12252 | 7190K|   58   (2)|00:00:01|
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| JSON_TAB              |     2 | 1182 |    1   (0)|00:00:01|
|   3 |    BITMAP CONVERSION TO ROWIDS       |                       |       |       |           |         |
|*  4 |     BITMAP INDEX SINGLE VALUE        | JSON_DATA_DELETED_IDX |       |       |           |         |
|   5 |   JSONTABLE EVALUATION               |                       |       |       |           |         |
--------------------------------------------------------------------------------------------------------
```

PERFORMANCE - B-TREE

```
SQL> select jt.json_data.OWNER,jt.json_data.OBJECT_NAME,
            jt.json_data.OBJECT_TYPE, jt.json_data.CREATED,jt.json_data.DELETED
       from json_tab jt where jt.json_data.OBJECT_NAME='ADD_JOB_HISTORY';

OWNER           OBJECT_NAME          OBJECT_TYP CREATED              DELETED
--------------- -------------------- ---------- -------------------- --------------------
HRREST          ADD_JOB_HISTORY      PROCEDURE  2017-03-02T08:43:54
HR              ADD_JOB_HISTORY      PROCEDURE  2017-03-02T08:41:44



-----------------------------------------------------------------------------------
| Id  | Operation               | Name     | Rows    | Bytes  | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |          |         |        | 2145K(100)|          |
|   1 |  NESTED LOOPS           |          |  6431K|  3667M| 2145K   (1)| 00:01:24 |
|   2 |   TABLE ACCESS FULL     | JSON_TAB | 78741  |    44M| 1912    (1)| 00:00:01 |
|   3 |   JSONTABLE EVALUATION |          |        |        |           |          |
-----------------------------------------------------------------------------------
```

PERFORMANCE - B-TREE

```
SQL> CREATE INDEX json_tab_objectname_idx ON json_tab jt (jt.json_data.OBJECT_NAME);

SQL> select jt.json_data.OWNER,jt.json_data.OBJECT_NAME,
            jt.json_data.OBJECT_TYPE, jt.json_data.CREATED,jt.json_data.DELETED
       from json_tab jt where jt.json_data.OBJECT_NAME='ADD_JOB_HISTORY';

OWNER            OBJECT_NAME          OBJECT_TYP CREATED              DELETED
---------------- -------------------- ---------- -------------------- --------------------
HRREST           ADD_JOB_HISTORY      PROCEDURE  2017-03-02T08:43:54
HR               ADD_JOB_HISTORY      PROCEDURE  2017-03-02T08:41:44

Plan hash value: 2179932252
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                              |Name                   |Rows  |Bytes | Cost (%CPU)|Time     |
|------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                       |      |      | 21575 (100)|         |
|   1 |  NESTED LOOPS                          |                       | 6431K| 4765M| 21575   (1)|00:00:01|
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED|JSON_TAB               |  787 |  591K|   204   (0)|00:00:01|
|*  3 |    INDEX RANGE SCAN                    |JSON_TAB_OBJECTNAME_IDX|  315 |      |     3   (0)|00:00:01|
|   4 |   JSONTABLE EVALUATION                 |                       |      |      |            |         |
-------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   3 - access("JT"."SYS_NC00047$"='ADD_JOB_HISTORY')
```

PERFORMANCE - B-TREE

```
SQL> CREATE INDEX json_tab_objectid_idx ON json_tab jt (jt.json_data.OBJECT_ID);



SQL> SELECT index_name , index_type, table_owner, table_name, table_type, uniqueness
        FROM user_indexes WHERE index_name = 'JSON_TAB_OBJECTID_IDX'

INDEX_NAME            INDEX_TYPE              TABLE_OWNER  TABLE_NAME   TABLE_TYPE   UNIQUENES
--------------------- ----------------------- ------------ ------------ ------------ ---------
JSON_TAB_OBJECTID_IDX FUNCTION-BASED NORMAL   NEIL         JSON_TAB     TABLE        UNIQUE



SQL> SELECT COLUMN_EXPRESSION FROM user_ind_expressions where index_name = 'JSON_TAB_OBJECTID_IDX';

COLUMN_EXPRESSION
-----------------
JSON_QUERY("JSON_DATA" FORMAT JSON , '$.OBJECT_ID' RETURNING
VARCHAR2(4000) ASIS WITHOUT ARRAY WRAPPER NULL ON ERROR)
```

PERFORMANCE - B-TREE

Index Constraints Are Upheld

```
CREATE UNIQUE INDEX json_tab_objectid_idx ON json_tab jt (jt.json_data.OBJECT_ID);



insert into json_tab(json_data) values ('{"OBJECT_ID":"79225"}');

insert into json_tab(json_data) values ('{"OBJECT_ID":"79225"}')
*
ERROR at line 1:
ORA-00001: unique constraint (JSON_TAB_OBJECTID_IDX) violated
```

PERFORMANCE - SEARCH INDEXES

- Search indexes are "Full-Text" or "Domain" Indexes tailored to JSON content
- They are "general" indexes, good for ad-hoc queries
- By default they are "update on commit"
  This may present a performance issue on insert
- Can indexes all of the fields in a JSON document along with their values
  including fields that occur inside arrays
- Can optimize any path-based search

PERFORMANCE - SEARCH INDEXES

```
CREATE SEARCH INDEX index-name ON table-name (json-column)
    FOR JSON PARAMETERS
  ('SEARCH_ON NONE| TEXT | TEXT_VALUE
    MEMORY memsize
    DATAGUIDE ON | OFF | ON CHANGE [ADD_VC|function]
    SYNC (manual | every 'interval' | on commit)')
```

```
SQL> CREATE SEARCH INDEX json_tab_sidx
         ON json_tab (json_data)
        FOR JSON PARAMETERS ('SEARCH_ON TEXT_VALUE DATAGUIDE ON')
```

PERFORMANCE - SEARCH INDEXES

```
SQL> SELECT jt.json_data.OWNER,jt.json_data.OBJECT_NAME,
            jt.json_data.OBJECT_TYPE, jt.json_data.CREATED,jt.json_data.DELETED
       FROM json_tab jt WHERE jt.json_data.OBJECT_NAME='ADD_JOB_HISTORY';


OWNER      OBJECT_NAM OBJECT_TYP CREATED    DELETED
---------- ---------- ---------- ---------- ----------
HR         ADD_JOB_HI PROCEDURE  2017-03-02
HRREST     ADD_JOB_HI PROCEDURE  2017-03-02

----------------------------------------------------------------------------------
| Id  | Operation                | Name       | Rows   | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |            |        |       |  2138K(100)|          |
|   1 |  NESTED LOOPS            |            |  6410K|  3961M|  2138K   (1)| 00:01:24 |
|   2 |   TABLE ACCESS FULL      | JSON_TAB | 78482  |    47M|  2083    (1)| 00:00:01 |
|   3 |   JSONTABLE EVALUATION |            |        |       |             |          |
----------------------------------------------------------------------------------
```

PERFORMANCE - SEARCH INDEXES

```
SQL> SELECT jt.json_data.OWNER,jt.json_data.OBJECT_NAME,
            jt.json_data.OBJECT_TYPE, jt.json_data.CREATED,jt.json_data.DELETED
       FROM json_tab jt WHERE
            JSON_VALUE(json_data, '$.OBJECT_NAME' RETURNING VARCHAR2(128) )='ADD_JOB_HISTORY'

OWNER      OBJECT_NAM OBJECT_TYP CREATED    DELETED
---------- ---------- ---------- ---------- ----------
HR         ADD_JOB_HI PROCEDURE  2017-03-02
HRREST     ADD_JOB_HI PROCEDURE  2017-03-02

-----------------------------------------------------------------------------------------------
| Id  | Operation                    | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |               |       |       |  1080 (100)|          |
|   1 |  NESTED LOOPS                |               |  3205 |  2071K|  1080   (1)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID| JSON_TAB      |    39 | 25350 |    17   (0)| 00:00:01 |
|*  3 |    DOMAIN INDEX              | JSON_TAB_SIDX |       |       |     4   (0)| 00:00:01 |
|   4 |   JSONTABLE EVALUATION       |               |       |       |            |          |
-----------------------------------------------------------------------------------------------
```

PERFORMANCE - SEARCH INDEXES

```
SQL> select json_data$description from json_view where json_data$description like '%xyz%';

JSON_DATA$DESCRIPTION
---------------------------------
yIsxyzjVKi EbM lnD xzC SEmMOwIXnO
OdvqTpnswt xyz XwY CYK ZMGATRPubM
uoxoPFweYu ynC GQO Ybl zgixyzPUEX
YeCRlWtAVv gRp Mjj xyz mhIbqBktMK
SArwACyLOV gmg GUV yWe XTxyzdITIC
NODxyzRzZS Gzp CDp Tcc qogTiqnVon
GcUOiRmfCG NUf uoL Ofn FJxyzOPhJX
nonJjlxdWt rim KwN ikW Rsxyzbxocd

------------------------------------------------------------------------------------
| Id  | Operation                 | Name     | Rows   | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |          |        |       |  2143K(100)|          |
|   1 |  NESTED LOOPS             |          |    32M|    19G|  2143K  (1)| 00:01:24 |
|   2 |   TABLE ACCESS FULL       | JSON_TAB | 78482  |    47M|  2083   (1)| 00:00:01 |
|   3 |   JSONTABLE EVALUATION    |          |        |       |            |          |
------------------------------------------------------------------------------------
```

PERFORMANCE - SEARCH INDEXES

```
SQL> SELECT jt.json_data.DESCRIPTION FROM json_tab jt
       WHERE JSON_TEXTCONTAINS (json_data, '$.DESCRIPTION', 'xyz');

DESCRIPTION
---------------------------------
CONVGWgwOd sHx XYZ vHt KqgDblxkLY
OdvqTpnswt xyz XwY CYK ZMGATRPubM
CRxGlqjmxo epM Xyz eTd jYgTiVWEte
nGdxiMBObi xPW Xyz iUG UgQaMCaYHi
SazHuMmHFE XYz lAC vFi HqGtBsijpe
YeCRlWtAVv gRp Mjj xyz mhIbqBktMK
CCCLuTNKbj Wuj xYZ EPz tKYiIRinQu
.
wdTnOECOLn xyZ RKG XpQ pTfvbWfwoZ

---------------------------------------------------------------------------------------------
| Id  | Operation                   | Name           | Rows   | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT            |                |        |       |    17 (100)|          |
|   1 |  TABLE ACCESS BY INDEX ROWID| JSON_TAB       |     39 | 25350 |    17   (0)| 00:00:01 |
|*  2 |   DOMAIN INDEX              | JSON_TAB_SIDX  |        |       |     4   (0)| 00:00:01 |
---------------------------------------------------------------------------------------------
```

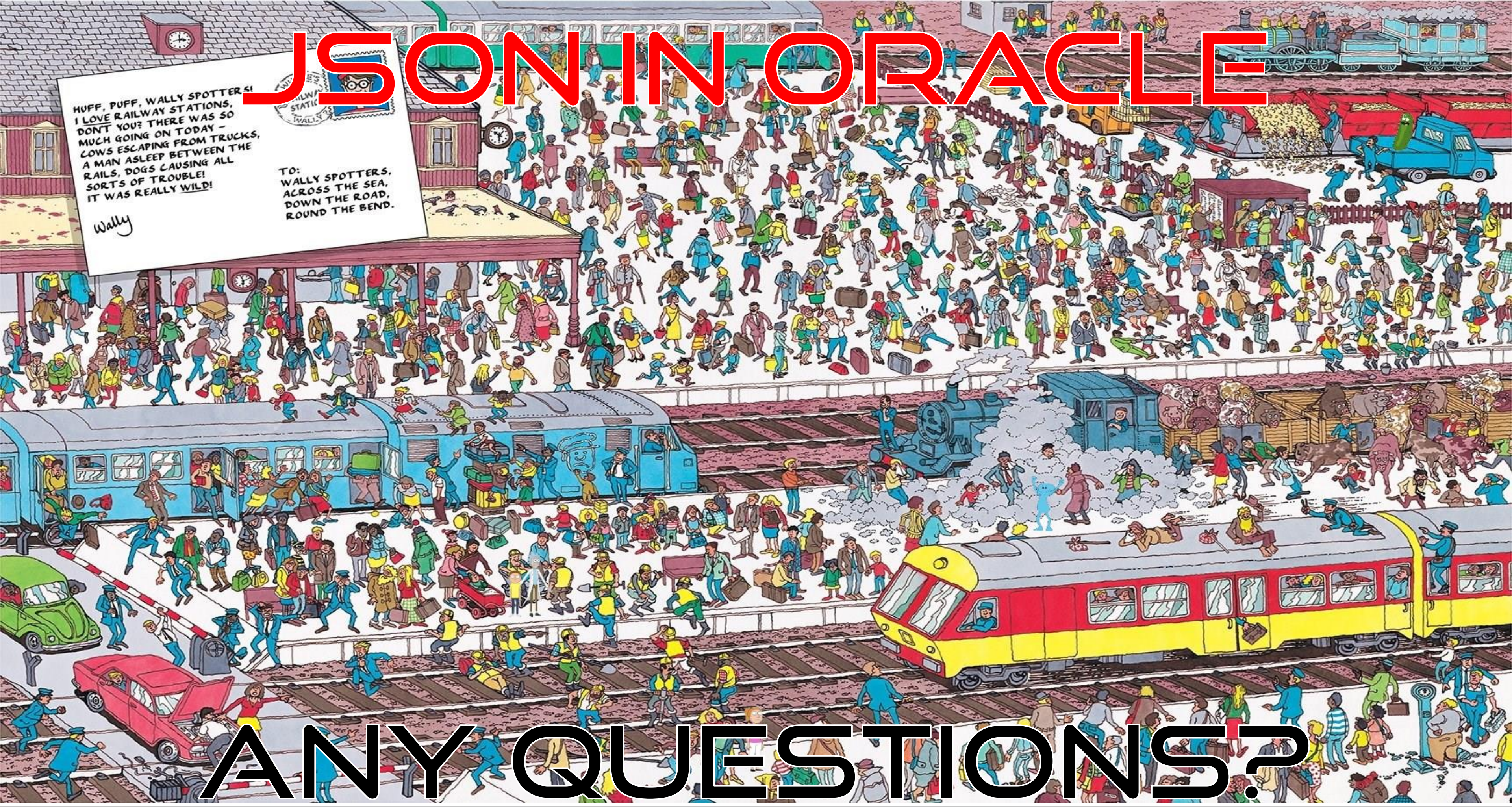
PERFORMANCE

- Pick the right index for the right job
- B-Tree Function-Based indexes are slightly slower then standard B-Tree to maintain
- Bitmap may cause contention/throughput issues
- Search Indexes are either out of date or will impact throughput with update-on-commit
- Table Scans are very heavy - JSONTABLE EVALUATION
  Put commonly accessed attributes at the start of the document

CONCLUSION

- You can store JSON effectively in Oracle
- Its not as fast as relational, but it's a lot more flexible
- You can expose it relationally, and it behaves like relational data (with some restrictions)
- Design it up front - at least the critical search areas!

JSON IN ORACLE


HUFF, PUFF, WALLY SPOTTERS! I LOVE RAILWAY STATIONS, DON'T YOU? THERE WAS SO MUCH GOING ON TODAY – COWS ESCAPING FROM TRUCKS, A MAN ASLEEP BETWEEN THE RAILS, DOGS CAUSING ALL SORTS OF TROUBLE! IT WAS REALLY WILD!
Wally
TO: WALLY SPOTTERS, ACROSS THE SEA, DOWN THE ROAD, ROUND THE BEND.
RAILWAY STATION
WALLY

ANY QUESTIONS?