

Dealing with JSON data in Oracle: DOs and DON'Ts

Beda Hammerschmidt
beda.hammerschmidt@oracle.com
Twitter: **bch_t**
Consulting Member of Technical Staff
Oracle, Database Group, JSON support
May 15th 2019, CERN, Geneva



**Live for
the Code**

ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved. | Confidential – Oracle Internal/Restricted/Highly Restricted

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Agenda

- 1 ➤ What is (good) JSON and when should I use it?
- 2 ➤ What's possible with a relational (Oracle) database?
- 3 ➤ Common use-cases, mistakes and best practices
- 4 ➤ NoSQL access to JSON, document stores
- 5 ➤ Questions

so,... what's JSON?

What's JSON?

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "age": 25  
}
```

What's JSON?

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "age": 25,  
  "address": {  
    "street": "21 2nd Street", "city": "New York",  
    "state": "NY", "postalCode": "10021",  
    "isBusiness": false  
  }  
}
```

What's JSON?

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "age": 25,  
  "address": {  
    "street": "21 2nd Street", "city": "New York",  
    "state": "NY", "postalCode": "10021",  
    "isBusiness": false  
  },  
  "phoneNumbers": [  
    { "type": "home", "number": "212 555-1234" },  
    { "type": "mobile", "number": "646 555-4567" }  
  ]  
}
```

What's JSON?

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "age": 25,  
  "address": {  
    "street": "21 2nd Street", "city": "New York",  
    "state": "NY", "postalCode": "10021",  
    "isBusiness": false  
  },  
  "phoneNumbers": [  
    {"type": "home", "number": "212 555-1234"},  
    {"type": "mobile", "number": "646 555-4567"}  
  ],  
  "bankruptcies": null,  
  "lastUpdated": "2019-05-13T13:03:35+0000"  
}
```

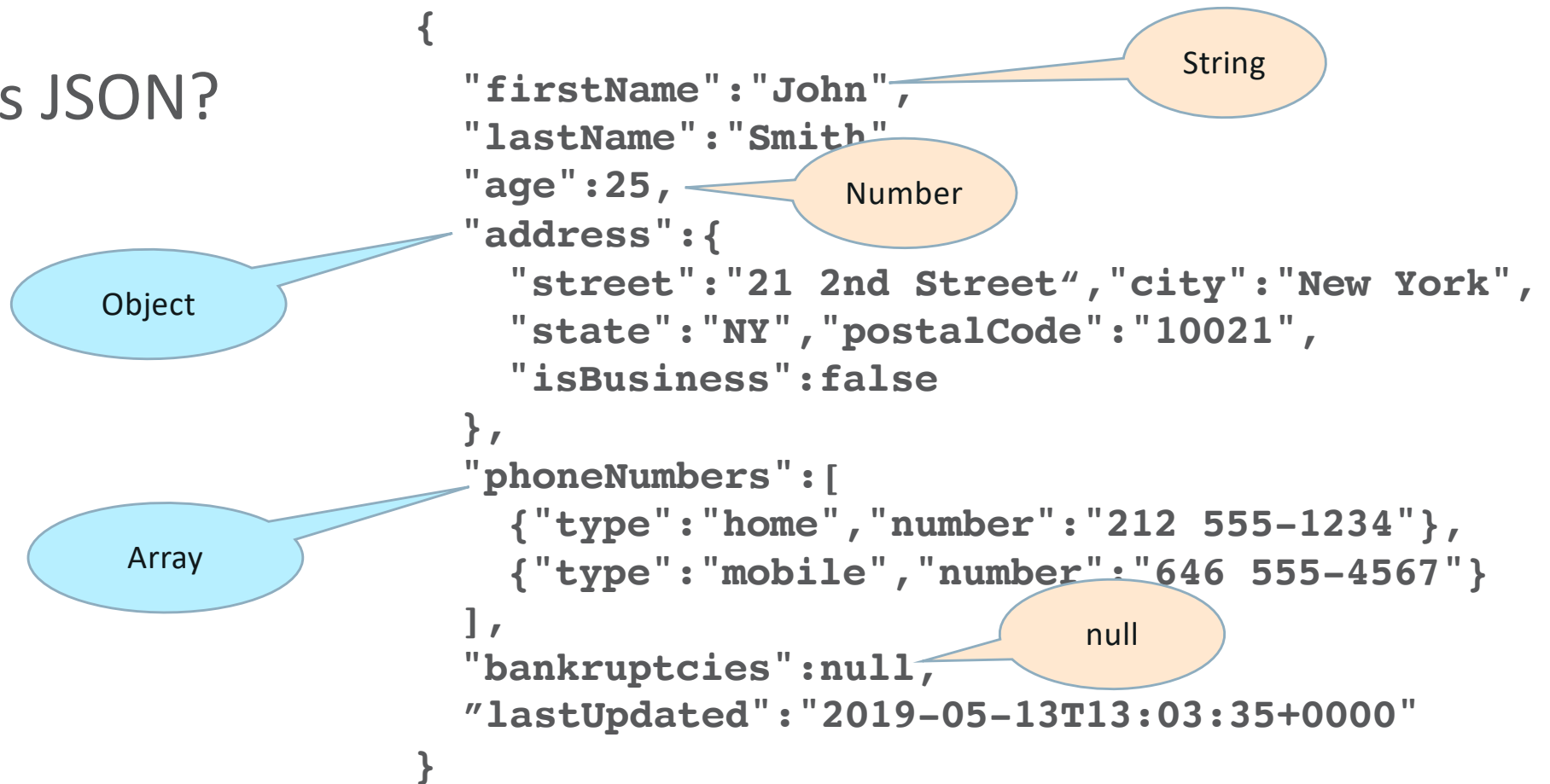
What's JSON?

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "age": 25,  
  "address": {  
    "street": "21 2nd Street", "city": "New York",  
    "state": "NY", "postalCode": "10021",  
    "isBusiness": false  
  },  
  "phoneNumbers": [  
    {"type": "home", "number": "212 555-1234"},  
    {"type": "mobile", "number": "646 555-4567"}  
  ],  
  "bankruptcies": null,  
  "lastUpdated": "2019-05-13T13:03:35+0000"  
}
```

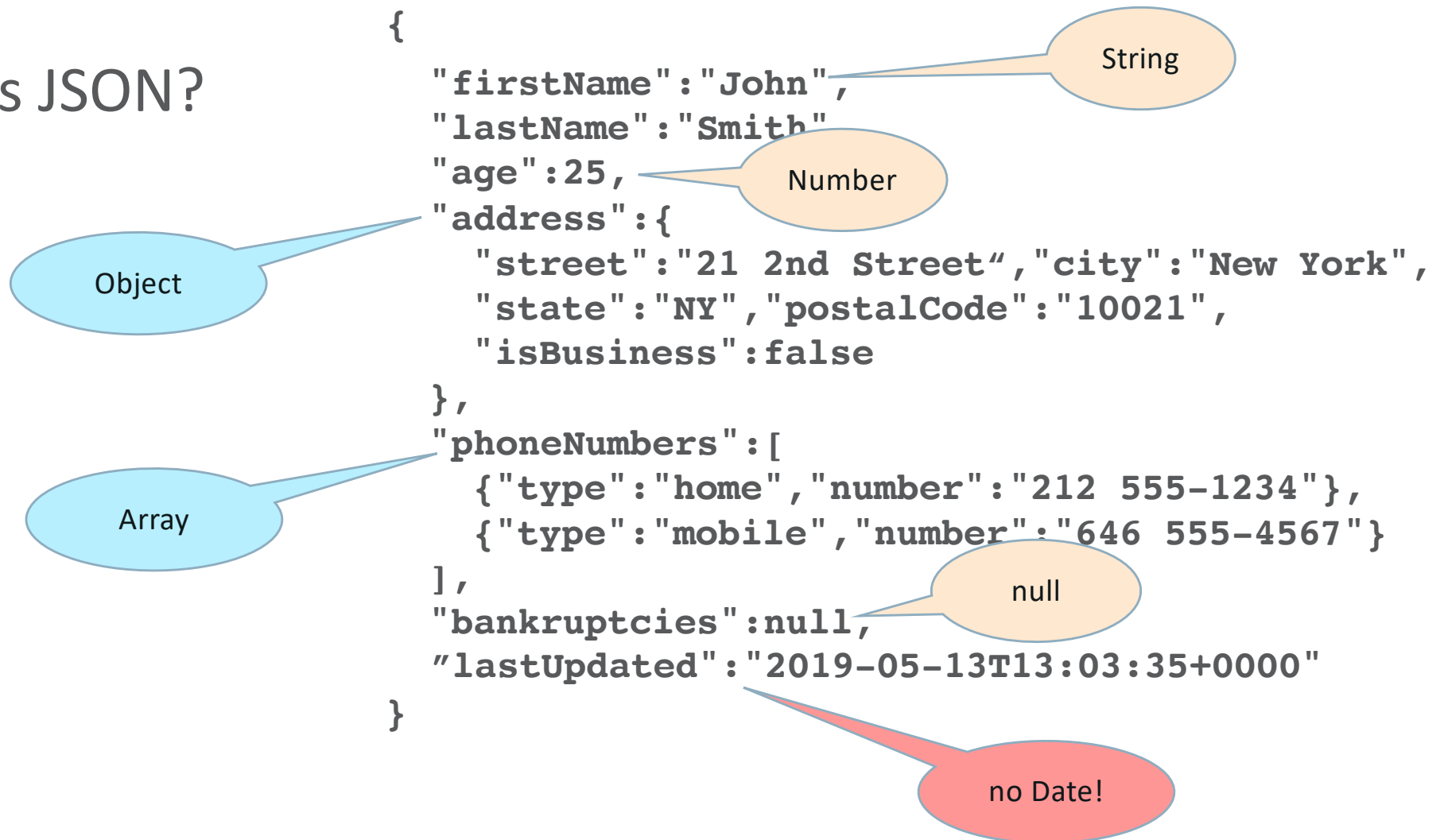
The diagram illustrates the data types of the JSON values using callouts:

- String**: Points to the value "John" in the `firstName` field.
- Number**: Points to the value 25 in the `age` field.
- Boolean**: Points to the value `false` in the `isBusiness` field.
- untyped null**: Points to the value `null` in the `bankruptcies` field.

What's JSON?



What's JSON?



What's JSON?

JSON vs XML :

- Both hierarchical
- JSON needs no schema
- Always UTF8-Unicode
- Fixed data types
- No type inheritance, or substitution groups
- no namespaces

....

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "street": "21 2nd Street", "city": "New York",
    "state": "NY", "postalCode": "10021",
    "isBusiness": false
  },
  "phoneNumbers": [
    { "type": "home", "number": "212 555-1234" },
    { "type": "mobile", "number": "646 555-4567" }
  ],
  "bankruptcies": null,
  "lastUpdated": "2019-05-13T13:03:35+0000"
}
```

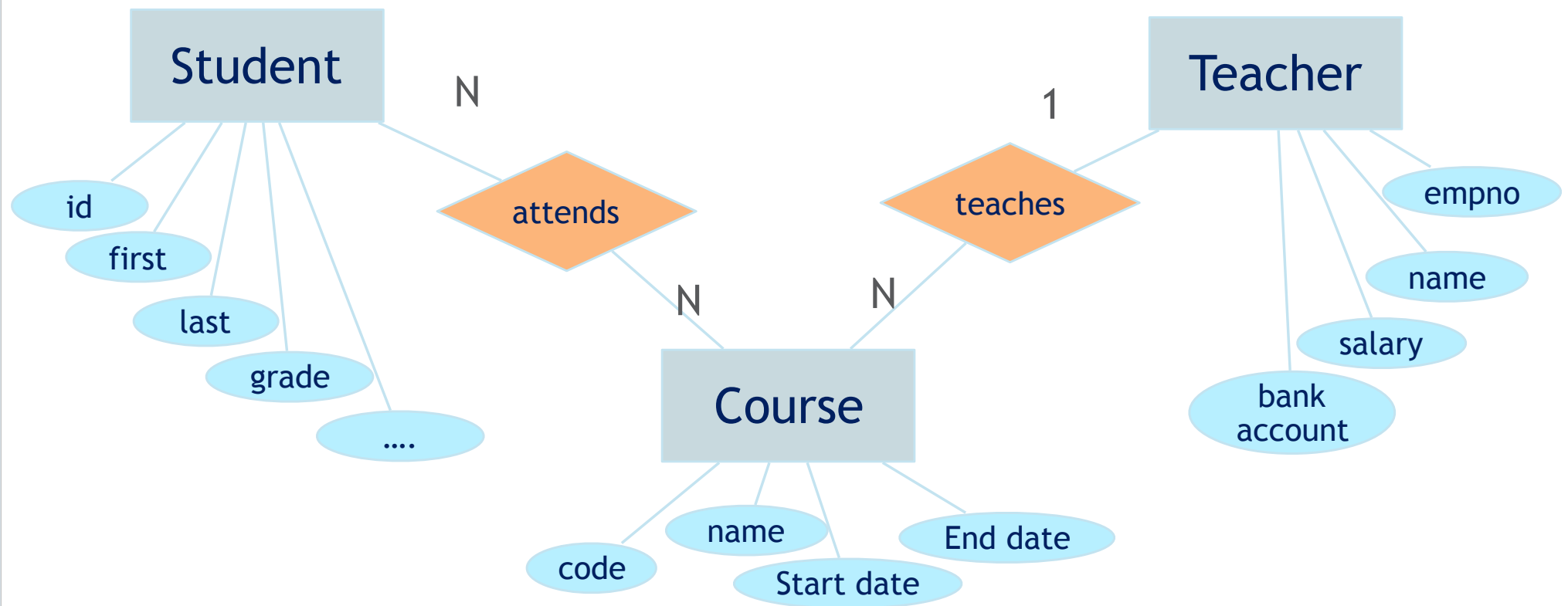
The schema-less fallacy



Sample Scenario: School/College



Entity Relationship Model (Peter Chen 1976)



Conversion to JSON



Conversion

How to find if a course is 'full'?
How can this be modeled?

Limited queriability:
How find all teachers of a student?

Inconsistencies possible if non-transactional.

Update all occurrences of same data

How to store a course without a teacher/student?

Data has to be repeated

```
{
  "id": "38728763872",
  "role": "Student",
  "user": {
    "id": "b7ed789f198a",
    "firstName": "Rick",
    "lastName": "Martin"
  },
  "course": {
    "dateRange": {
      "startDate": ISODate("2011-12-30T07:00:00Z"),
      "endDate": ISODate("2012-01-30T07:00:00Z")
    },
    "courseId": "734234274",
    "code": "MATH/101",
    "title": "Introduction to Mathematics"
  }
}
```

Figure 2: Our Simplified MongoDB Data Model

Conversion

How to find if a course is 'full'?
How can this be modeled?

Limited queriability:
How find all teachers of a student?

Inconsistencies possible if non-transactional.

Update all occurrences of same data

This is 1966
IBM IMS
hierarchical data model!
We just embarked on a
50yr time travel!

How to store a course without a teacher/student?

Data has to be repeated

Figure 2: Our Simplified MongoDB Data Model



JSON support in the Oracle Database

SQL/JSON -> SQL 2016

JSON Support in the Oracle database

- JSON storage

```
create table emp (  
    empno    number,  
    first    varchar2(50),  
    last     varchar2(50),  
    salary   number,  
    flex     clob,  
    check (flex IS JSON));
```

```
insert into emp values( 1, 'Jon', 'Smith', 2000,  
    '{"skills":["Java", "C", "C++"],  
    "photoUrl":"img/eydh763s.jpg"}');
```

```
insert into emp values( 2, 'Amanda', 'Jones', 2500,  
    '{"skills":["JavaScript","nodeJs"]}');
```

JSON Support in the Oracle database

- JSON storage

```
create table emp (  
    empno number,  
    first varchar2(50),  
    last varchar2(50),  
    salary number,  
    flex clob,  
    check (flex IS JSON));
```

Oracle performance tip:

JSON is always UTF8

- Use 'al32utf8' character set if possible
- Use BLOB instead of CLOB
CLOB stores data as UCS2/UTF16
- Use lob data api to insert/fetch data
 - getBytes/setBytes instead of
 - getBlob/setBlob!

```
emp values( 1, 'Jon', 'Smith', 2000,  
    ': ["Java", "C", "C++"]',  
    'url': "img/eydh763s.jpg" }');
```

```
emp values( 2, 'Amanda', 'Jones', 2500,  
    ': ["JavaScript", "nodeJs"] }');
```

JSON Support in the Oracle database

- JSON storage
- JSON value extraction

```
select e.flex.skills  
from emp e  
where e.flex.skills like '%Java%';
```

	SKILLS
1	["Java","C","C++"]
2	["JavaScript","nodeJs"]

JSON Support in the Oracle database

- JSON storage
- JSON value extraction

```
select e.flex.skills  
from emp e  
where e.flex.skills like '%Java%';
```

	SKILLS
1	["Java","C","C++"]
2	["JavaScript","nodeJs"]

```
select JSON_VALUE (flex,'$.photoUrl'  
                  DEFAULT 'img/default.jpg' on empty)  
from emp;
```

	JSON_VALUE(E.FLEX,'\$.PHOTOURL'
1	img/eydh763s.jpg
2	img/default.jpg

JSON Support in the Oracle database

- JSON storage
- JSON value extraction

```
select e.flex.skills
from emp e
where e.flex.skills like '%Java%';
```

	SKILLS
1	["Java","C","C++"]
2	["JavaScript","nodeJs"]

Oracle functionality tip:

SQL/JSON operators have default for simple usage – but also many options:

JSON_VALUE(flex, '\$.path' RETURNING NUMBER(4,2),
ERROR ON ERROR
NULL ON EMPTY)

Default return type is VARCHAR2(4000)

```
flex, '$.photoUrl'  
DEFAULT 'img/default.jpg' on empty)
```

	JSON_VALUE(E.FLEX,'\$.PHOTOURL'
1	img/eydh763s.jpg
2	img/default.jpg

JSON Support in the Oracle database: Predicates

- JSON storage
- JSON value extraction

```
select e.flex.skills  
from emp e  
where e.flex.skills like '%Java%';
```

	SKILLS
1	["Java","C","C++"]
2	["JavaScript","nodeJs"]

```
select JSON_QUERY(flex, '$.skills')  
from emp  
where JSON_EXISTS (flex, '$.skills?(@ == "Java")');
```

	JSON_QUERY(FLEX,'\$.SKILLS')
1	["Java","C","C++"]

JSON Support in the Oracle database

- JSON storage
- JSON value extraction
- JSON table projection

JSON_TABLE: 'flatten' the JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "street": "21 2nd Street",
    "city": "New York",
    "state": "NY", "postalCode": "10021",
    "isBusiness": false
  },
  "phoneNumbers": [
    {"type": "home",
     "number": "212 555-1234"},
    {"type": "mobile",
     "number": "646 555-4567"}
  ],
  "bankruptcies": null,
  "lastUpdated": "2019-05-13T13:03:35"
}
```

JSON_TABLE: 'flatten' the JSON data

```
Select id, jt.* from
  custData,
  JSON_TABLE(jcol, '$' columns (
    "FIRST"          path '$.firstName',
    "AGE"            number          path '$.age',
    "STATE"          varchar2(2)     path '$.address.state',
    "PHONE"          FORMAT JSON     path
                                '$.phoneNumbers.number'
                                with array wrapper
  )) jt;
```

JSON_TABLE: 'flatten' the JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "street": "21 2nd Street",
    "city": "New York",
    "state": "NY", "postalCode": "10021",
    "isBusiness": false
  },
  "phoneNumbers": [
    { "type": "home",
      "number": "212 555-1234" },
    { "type": "mobile",
      "number": "646 555-4567" }
  ],
  "bankruptcies": null,
  "lastUpdated": "2019-05-13T13:03:35"
}
```

JSON_TABLE: 'flatten' the JSON data

```
Select id, jt.* from
  custData,
  JSON_TABLE(jcol, '$' columns (
    "FIRST"          path '$.firstName',
    "AGE"             number          path '$.age',
    "STATE"           varchar2(2)     path '$.address.state',
    "PHONE"           FORMAT JSON     path
                                '$.phoneNumbers.number'
                                with array wrapper
  )) jt;
```

ID	FIRST	AGE	STATE	PHONE
----	-------	-----	-------	-------

--	--	--	--	--

JSON_TABLE: 'flatten' the JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "street": "21 2nd Street",
    "city": "New York",
    "state": "NY", "postalCode": "10021",
    "isBusiness": false
  },
  "phoneNumbers": [
    {"type": "home",
     "number": "212 555-1234"},
    {"type": "mobile",
     "number": "646 555-4567"}
  ],
  "bankruptcies": null,
  "lastUpdated": "2019-05-13T13:03:35"
}
```

JSON_TABLE: 'flatten' the JSON data

```
Select id, jt.* from
  custData,
  JSON_TABLE(jcol, '$' columns (
    "FIRST"          path '$.firstName',
    "AGE"            number          path '$.age',
    "STATE"          varchar2(2)     path '$.address.state',
    "PHONE"          FORMAT JSON     path
                                '$.phoneNumbers.number'
                                with array wrapper
  )) jt;
```

ID	FIRST	AGE	STATE	PHONE
1	John	25	NY	["212 555-1234", "646 555-4567"]

JSON_TABLE: 'flatten' the JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "street": "21 2nd Street",
    "city": "New York",
    "state": "NY", "postalCode": "10021",
    "isBusiness": false
  },
  "phoneNumbers": [
    {"type": "home",
     "number": "212 555-1234"},
    {"type": "mobile",
     "number": "646 555-4567"}
  ],
  "bankruptcies": null,
  "lastUpdated": "2019-05-13T13:03:35"
}
```

JSON_TABLE: 'flatten' the JSON data

```
Select id, jt.* from
  custData,
  JSON_TABLE(jcol, '$' columns (
    "FIRST"          path '$.firstName',
    "AGE"            number          path '$.age',
    "STATE"          varchar2(2)     path '$.address.state',
    NESTED PATH '$.phoneNumbers[*]' columns (
      "PHONES"       path '$.number
    )
  )) jt;
```

ID	FIRST	AGE	STATE	PHONES
1	John	25	NY	212 555-1234
1	John	25	NY	646 555-4567

JSON_TABLE: 'flatten' the JSON

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "street": "21 2nd Street",
    "city": "New York",
    "state": "NY", "postalCode": "10021",
    "isBusiness": false
  },
  "phones": [
    {
      "number": "212 555-1234"
    },
    {
      "number": "646 555-4567"
    }
  ]
}
```

JSON_TABLE: 'flatten' the JSON data

```
Select id, jt.* from
  custData,
  JSON_TABLE(jcol, '$' columns (
    "FIRST"          path '$.firstName',
    "AGE"             number          path '$.age',
    "STATE"           varchar2(2)     path '$.address.state',
    "PHONES"          PATH '$.phoneNumbers[*]' columns (
      "number"         path '$.number'
```

Oracle performance tip:

JSON_TABLE columns can have different semantics:
JSON_VALUE, JSON_QUERY, JSON_EXISTS
all options supported

Write on JSON_TABLE with multiple columns instead of
SELECT JSON_VALUE, JSON_VALUE, JSON_QUERY
FROM ...
WHERE JSON_EXISTS

STATE	PHONES
NY	212 555-1234
NY	646 555-4567

JSON Support in the Oracle database

- JSON storage
- JSON value extraction
- JSON table projection
- JSON generation

JSON Generation

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

```
Select JSON_OBJECT(  
    'id' VALUE empno,  
    'name' VALUE ename )  
from emp;
```

```
{"id":7969,"name":"SMITH"}
```

```
{"id":7499,"name":"ALLEN"}
```

```
{"id":7521,"name":"WARD"}
```

```
{"id":7566,"name":"JONES"}
```

JSON Generation: Aggregates

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

```
Select JSON_ARRAYAGG(EMPNO)
from emp;
```

```
[7369, 7499, 7521, 7566, ....]
```

JSON Generation: Departments with all employees

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

```
Select JSON_OBJECTAGG(  
  dname VALUE  
  
) from dept d;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

JSON Generation: Departments with all employees

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

```
Select JSON_OBJECTAGG(  
  dname VALUE ..
```

```
) from dept d;
```

```
{  
  "ACCOUNTING":...,  
  "RESEARCH":...,  
  "SALES":...,  
  "OPERATIONS":...  
  .....  
}
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

JSON Generation: Departments with all employees

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

```
Select JSON_OBJECTAGG(  
  dname VALUE select JSON_ARRAYAGG(...  
  ) from emp e where e.deptno = d.deptno  
) from dept d;
```

```
{  
  "ACCOUNTING":[...],  
  "RESEARCH"[..],  
  "SALES":[...],  
  "OPERATIONS":[...]  
}  
  
.....
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

JSON Generation: Departments with all employees

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

```
Select JSON_OBJECTAGG(  
  dname VALUE select JSON_ARRAYAGG(  
    JSON_OBJECT('name' VALUE ename)  
  ) from emp e where e.deptno = d.deptno  
) from dept d;
```

```
{  
  "ACCOUNTING": [{  
    "name": "CLARK"  
  }, {  
    "name": "KING"  
  }, {  
    "name": "MILLER"  
  }],  
  "RESEARCH": [{  
    "name": "SMITH"  
  }, {  
    "name": "JONES"  
  }, {.....  
  }.....  
}
```

JSON Generation: Departments with all employees

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10

```
Select JSON_OBJECTAGG(  
  dname VALUE select JSON_ARRAYAGG(  
    JSON_OBJECT('name' VALUE ename)  
  ) from emp e where e.deptno = d.deptno  
) from dept d;
```

```
{  
  "ACCOUNTING": [{  
    "name": "CLARK"  
  }, {  
    "name": "KING"  
  }, {  
    "name": "MILLER"  
  }],  
  "RESEARCH": [{  
    "name": "SMITH"  
  }, {  
    "name": "JONES"  
  }, {  
    .....  
  }],  
  .....  
}
```

Oracle performance tip:

total output size may require lob data type
(use RETURNING CLOB clause)

intermediate results may not be as big
try to avoid intermediate lobes if possible

Varchar2 goes up to 32767 bytes

JSON Support in the Oracle database

- JSON storage
- JSON value extraction
- JSON table projection
- JSON generation
- JSON Indexing

Functional index for one path: B-Tree

```
create index cost_idx on emp (  
    json_value(flex, '$.costcenter'  
        returning number  
        error on error  
        null on empty));
```

Search Index (index all): posting list

```
create search index search_idx  
    on emp (flex) for JSON;  
(Search index allows full text search, e.g. contains)
```

JSON Support in the Oracle database

- JSON storage
- JSON value extraction
- JSON table projection
- JSON generation
- JSON Indexing
- JSON Dataguide

```
select JSON_DATAGUIDE(e.flex)  
from emp e;
```

JSON Support in the Oracle database

- JSON storage
- JSON value extraction
- JSON table projection
- JSON generation
- JSON Indexing
- JSON Dataguide

```
select JSON_DATAGUIDE  
from emp e;
```

```
[{  
  "o:path": "$.skills",  
  "type": "array",  
  "o:length": 32  
}, {  
  "o:path": "$.skills[*]",  
  "type": "string",  
  "o:length": 16  
}, {  
  "o:path": "$.photoUrl",  
  "type": "string",  
  "o:length": 16  
}]
```


Data Guide: Relational access to JSON content

```
call DBMS_JSON.CREATE_VIEW_ON_PATH(  
    'THEATER_VIEW',  
    'THEATER',  
    'JSON_DOCUMENT',  
    '$.id'  
)
```

- Automatically create a relational view of your JSON content
 - Views are based on JSON_TABLE operator
- Use the PATH argument to control which keys are included in the view
- Automatically generates unique column names

```
desc THEATER_VIEW
```

Name	Null?	Type
-----	-----	-----
ID	NOT NULL	VARCHAR2 (255)
CREATED_ON	NOT NULL	TIMESTAMP (6)
LAST_MODIFIED	NOT NULL	TIMESTAMP (6)
VERSION	NOT NULL	VARCHAR2 (255)
JSON_DOCUMENT\$id		NUMBER
JSON_DOCUMENT\$name		VARCHAR2 (64)
JSON_DOCUMENT\$city		VARCHAR2 (32)
JSON_DOCUMENT\$state		VARCHAR2 (2)
JSON_DOCUMENT\$street		VARCHAR2 (64)
JSON_DOCUMENT\$zipCode		VARCHAR2 (8)
JSON_DOCUMENT\$phoneNumber		VARCHAR2 (4)

```
select count(*) COUNT  
  from THEATER_VIEW  
 where "JSON_DOCUMENT$zipCode" = 94115
```

```
      COUNT  
-----  
          3
```

Data Guide: Capturing changes to the structure of your JSON

```
CREATE INDEX SCREENING_SEARCH
ON "Screening" (JSON_DOCUMENT) FOR JSON
PARAMETERS ('SEARCH_ON NONE
DATAGUIDE ON
CHANGE LOG_JSON_CHANGES')
```

```
select JSON_PATH, JSON_TYPE, USERID, TIMESTAMP
from JSON_CHANGE_LOG
```

JSON_PATH	JSON_TYPE	USERID	TIMSTAMP
\$.movieId	3	STUDENT01	2017-02-05T14:57:37Z
\$.screenId	3	STUDENT01	2017-02-05T14:57:37Z
\$.startTime	4	STUDENT01	2017-02-05T14:57:37Z
\$.theaterId	3	STUDENT01	2017-02-05T14:57:37Z
\$.ticketPricing	5	STUDENT01	2017-02-05T14:57:37Z
\$.ticketPricing.adultPrice	3	STUDENT01	2017-02-05T14:57:37Z
\$.ticketPricing.childPrice	3	STUDENT01	2017-02-05T14:57:37Z
\$.ticketPricing.seniorPrice	3	STUDENT01	2017-02-05T14:57:37Z
\$.seatsRemaining	3	STUDENT01	2017-02-05T14:57:37Z

9 rows selected.

- Create a data guide enabled search index
 - “SEARCH_ON NONE” prevents the index functioning as a search index
 - Attach the procedure to the index
- The change procedure is called once for each new path found while building the index
- The change procedure is called every time a new path is found during insert and update operations

JSON Support in the Oracle database

- JSON storage
- JSON value extraction
- JSON table projection
- JSON generation
- JSON Indexing
- JSON Dataguide
- JSON Cross Functional

Oracle uses a binary **in-memory** representation for JSON:
Post-parse format allowing 'jump' navigation.

Exadata pushdown pf predicates is done for JSON_VALUE, JSON_EXISTS and JSON_QUERY with WHERE CLAUSE.
Data/lob size limit 4k!

Partitioning possible on JSON value but preferably done on relational columns.

Security features like Encryption, VPD, Redaction, ... usable with JSON data

JSON Support in the Oracle database

- JSON storage
- JSON value extraction
- JSON table projection
- JSON generation
- JSON Indexing
- JSON Dataguide
- JSON Cross Functional
- JSON in PL/SQL

```
create or replace function addSkill(skills varchar2,  
                                   newSkill varchar2)  
    return varchar2 is  
  
    skillsArr JSON_ARRAY_T;  
    result    varchar2(4000);  
begin  
    skillsArr := JSON_ARRAY_T(skills);  
    skillsArr.append(newSkill);  
    result := skillsArr.stringify;  
    return result;  
end;  
/  
  
select addSkill(['C++'], 'Java') from dual;
```

	ADDSKILL(['C++'],'JAVA')
1	["C++","Java"]

JSON Support in the Oracle database

- JSON storage
- JSON value extraction
- JSON table projection
- JSON generation
- JSON Indexing
- JSON Dataguide
- JSON Cross Functional
- JSON in PL/SQL
- JSON Updates
- JSON_TRANSFORM in progress
 - In standardization process
 - Declarative Updates
- Available today:
 - PL/SQL programmatic updates (12.2)
 - JSON_MERGEPATCH (18c)
 - Update entire document (12.1.0.1)

Common use cases

Common JSON use cases

- 1) JSON Ingestion to relational application (import)
 - Examples: Google Clickstream, travel bookings, tax-filing, social media feeds,...
 - Often no control over JSON structure (schema). Sometimes originating from XML
 - JSON_TABLE to 'shred' relational data, (materialized) views
 - Often JSON is not only imported but also stored and queried (schema-less storage)
- 2) JSON Generation (Export)
 - Export of data and message generation
 - Often from relational (legacy) applications
 - Example: modern JSON-based UI on top of 'old' and working application
 - Sometimes PL/SQL used to generate complex (conditional) messages
 - Example: Oracle RAC autonomous health framework: node status as JSON record

Common JSON use cases

- 3) Flexfields (hybrid storage)
 - Core of application is relational but ‘new’ fields are in a (flat) JSON column
 - Schema flexibility, easier/faster to extend application
 - Data augmentation by third parties: fields are unknown and never defined,
 - JSON Search index (index all values)
- 4) Sparse columns (Avoid the 1000 column limits)
 - metadata of financial assets (>1000 values), more added dynamically
 - Storage in multiple JSON varchar2(4000) columns
 - Exadata pushdown on JSON_EXISTS queries, no index
 - JSON_TABLE views, to select relevant values as columns (<1000 cols per view)

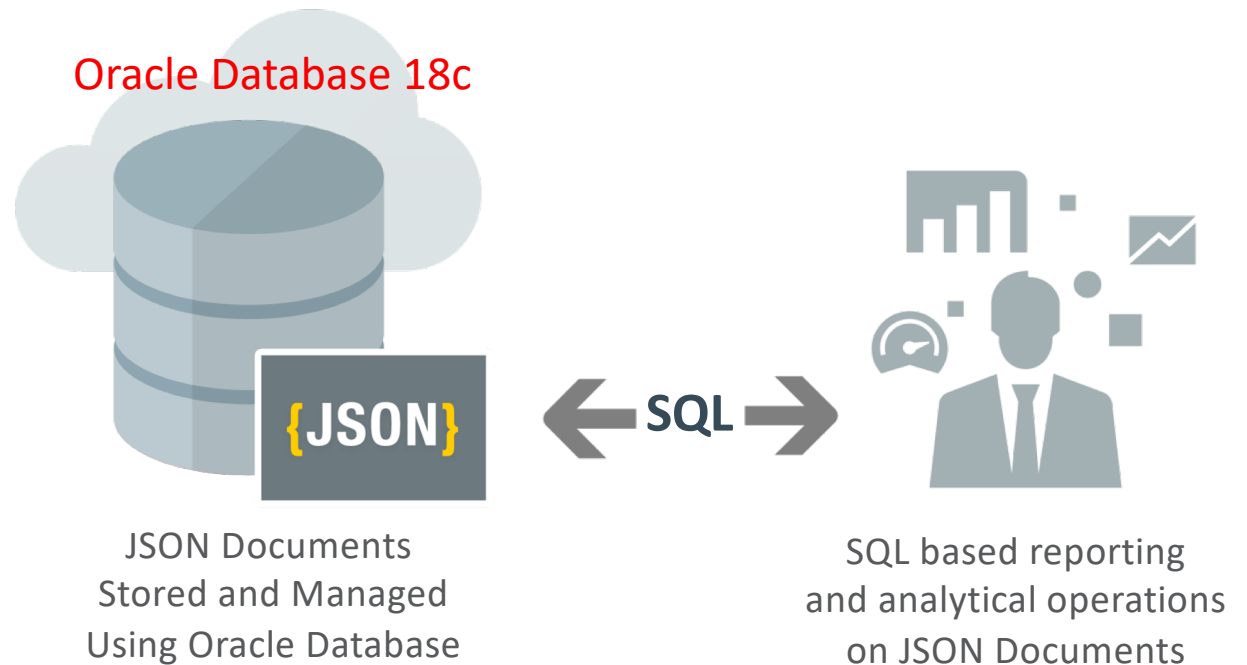
Common JSON use: Oracle as a document store

- 5) Document-Store:
 - Entire data-model is JSON based - similar to MongoDB
 - Example: point of sales (super market; consumer electronics vendor)
 - Every sale/order become a JSON document (1Kb to 100s of Kbs, average under 5kb)
 - Why JSON and not relational?
 - Perception: JSON is modern, productivity, developers want No-SQL, "today it's done this way"
 - Why Oracle?
 - Richer feature- set: e.g. Transactions, Joins, Views, Generation (Reporting)
 - Co-Existence of JSON and non-JSON data (multi-model DB)
 - DBA skills: apply existing skills and experience to JSON
- Simple Oracle Document Access (SODA) caters to this use case

Simple Oracle Document Access (SODA)

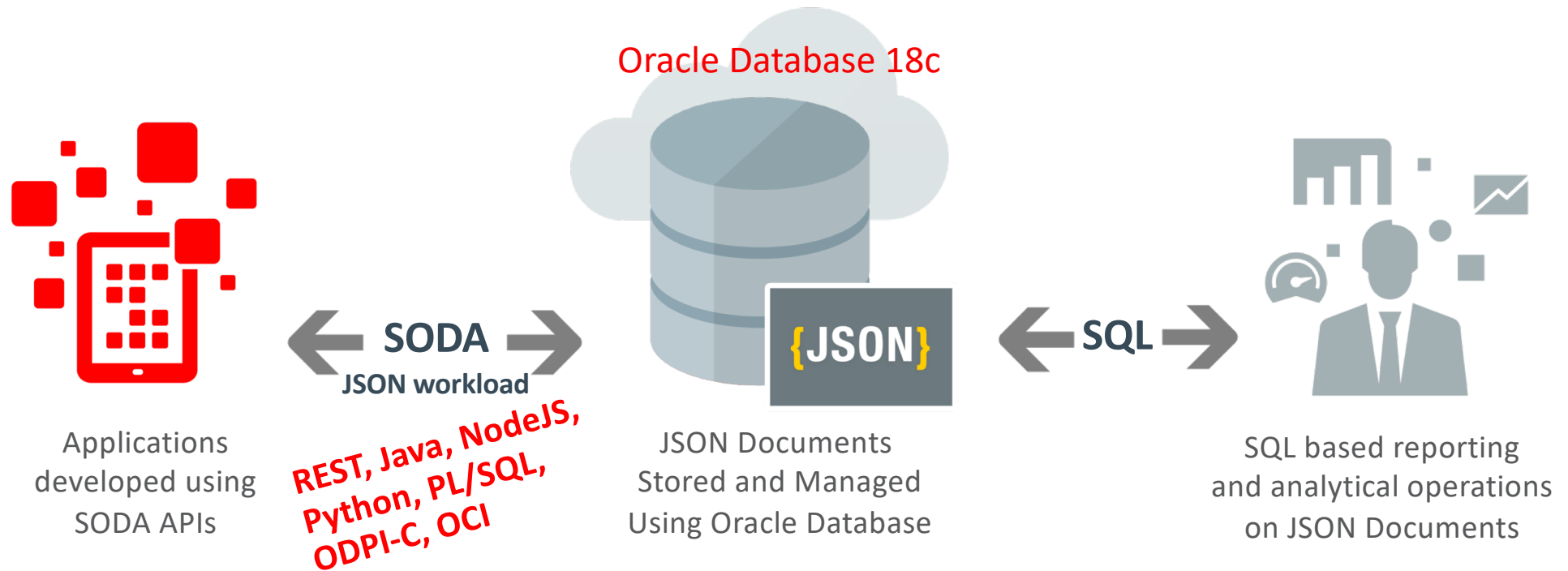
Conclusion: JSON storage and easy access Api(SODA)

Simple NoSQL Development experience



Conclusion: JSON storage and easy access Api(SODA)

Simple NoSQL Development experience



Document Store Model

- A **document** is a JSON (or binary) instance
 - Document is identified by a key (get, update, delete by key)
- Store documents inside a ***collection***
 - Collection stores similar documents
 - Collection is identified by name
- One or more collections reside in a ***database***
 - Application connects to database

Document Store Model mapped to relational model

- A **document** is a JSON (or binary) instance
 - Document is identified by a key (get, update, delete by key)
- Store documents inside a ***collection***
 - Collection stores similar documents
 - Collection is identified by name
- One or more collections reside in a ***database***
 - Application connects to database

Document Store Model mapped to relational model

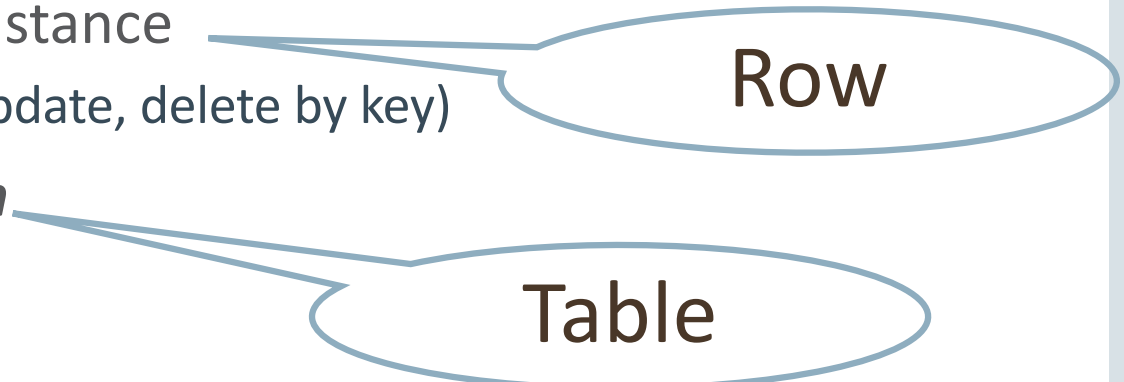
- A **document** is a JSON (or binary) instance
 - Document is identified by a key (get, update, delete by key)
- Store documents inside a **collection**
 - Collection stores similar documents
 - Collection is identified by name
- One or more collections reside in a **database**
 - Application connects to database

A blue oval containing the word 'Table' is connected by a blue line to the word 'collection' in the list above. This visualizes the mapping between a document store collection and a relational database table.

Table

Document Store Model mapped to relational model

- A **document** is a JSON (or binary) instance
 - Document is identified by a key (get, update, delete by key)
- Store documents inside a **collection**
 - Collection stores similar documents
 - Collection is identified by name
- One or more collections reside in a **database**
 - Application connects to database



Row

Table

Document Store Model mapped to relational model

- A **document** is a JSON (or binary) instance
 - Document is identified by a key (get, update, delete by key)
- Store documents inside a **collection**
 - Collection stores similar documents
 - Collection is identified by name
- One or more collections reside in a **database**
 - Application connects to database

Row

Table

Schema

Document Store Model mapped to relational model

- A **document** is a JSON (or binary) instance
 - Document is identified by a key (get, update, delete by key)
- Store documents inside a **collection**
 - Collection stores similar documents
 - Collection is identified by name
- One or more collections reside in a **database**
 - Application connects to database

Row

Table

Schema

KEY	JSON_DOC	VERSION	CREATED_ON	LAST_MODIFIED
3C990A8D39...	{"name" : "Alex"}	75611B292...	2018-09-12T00:00:00Z	2018-09-12T00:00:00Z

Creating and dropping collections

Node.js

```
conn = await oracledb.getConnection(...);  
db = conn.getSodaDatabase();  
coll = await db.createCollection("sweets");  
await coll.drop();
```

Java

```
OracleClient cl = new OracleRDBMSClient();  
db = cl.getDatabase(jdbcConn);  
  
OracleCollection coll =  
db.admin().createCollection("sweets");  
coll.admin().drop();
```

Python

```
conn = cx_Oracle.connect(...);  
db = conn.getSodaDatabase();  
coll = db.createCollection("sweets");  
coll.drop();
```

PL/SQL

```
coll := dbms_soda.create_collection('sweets');  
  
select dbms_soda.drop_collection('sweets') from  
dual;
```

Inserting, finding, updating documents

Node.js *(other implementations have similar syntax)*

- `resultDoc = await coll.insertOneAndGet({name : "Alexander"});`
- `key = resultDoc.key;`
- `col.find().key("k1");`
- `col.find().key("k1"). replaceOne(newDoc);`
- `col.find().filter({"name" : "alex"});`
- `col.find().filter({"name" : "alex"}).remove();`
- `col.find().filter({"name" : "chris"}).skip(10).limit(10).getCursor();`

QBEs can be sophisticated!

```
{ "$query" : {  
    "empno" : { "$lte": 10041 },  
    "name" : { "$startsWith" : "Melissa" },  
    "salary" : { "$gt" : 200000 },  
    "title" : { "$contains" : "President" },  
},  
"$orderby" : [  
    {"path" : "salary", "order" : "asc"},  
    {"path" : "department.name", "order" : "desc"}  
]  
}
```

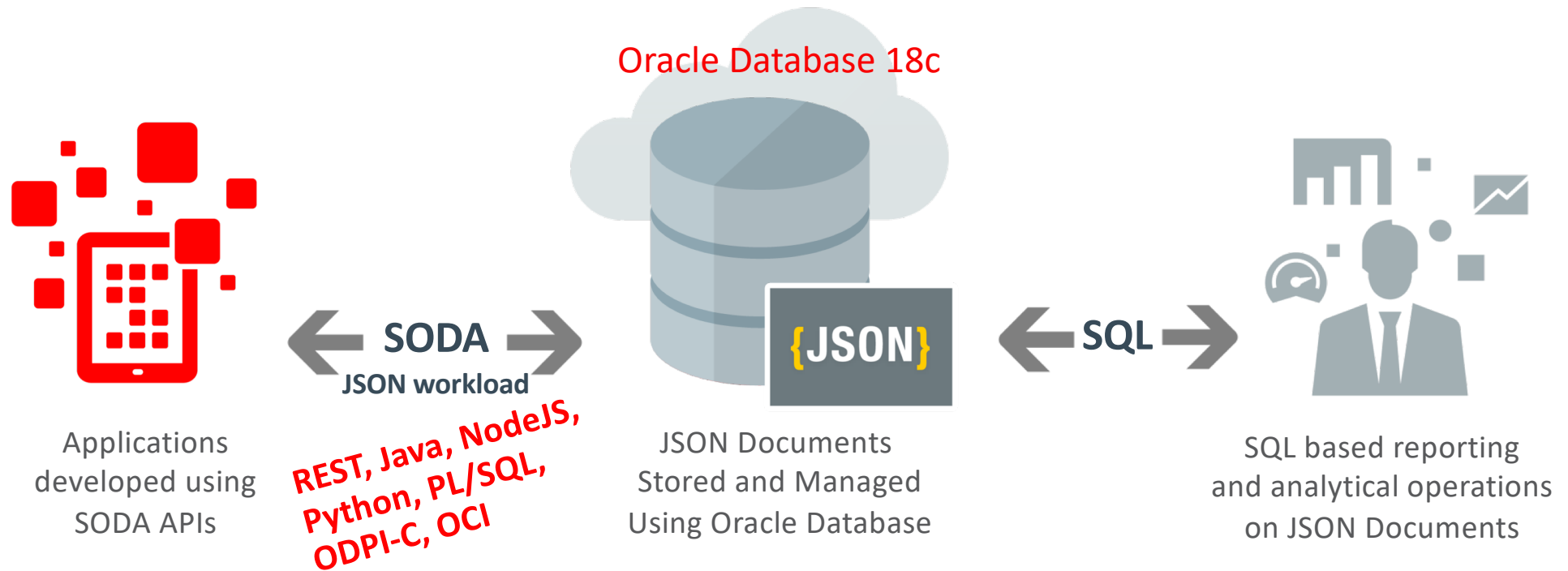
QBE rewritten to SQL

```
SELECT "ID", "JSON_DOCUMENT" FROM "Employees" coll
WHERE JSON_Exists(coll."JSON_DOCUMENT",
    '$?(@.empno <= $B0 &&
        @.salary > $B1 &&
        @.name starts with $B2)')
    PASSING 10041 AS "B0", 200000 AS "B1", 'Melissa' AS "B2")
AND JSON_TextContains(coll."JSON_DOCUMENT",
    '$.title', 'President')
ORDER BY JSON_Value(coll."JSON_DOCUMENT", '$.salary') ASC,
    JSON_Value(coll."JSON_DOCUMENT", '$.department.name') DESC;
```

Conclusion

Conclusion: JSON storage and easy access Api(SODA)

Simple NoSQL Development experience



JSON tutorial on LiveSQL

 **ORACLE** Live SQL

Feedback Help maria.colgan@oracle.com

 Home

 SQL Worksheet

 My Session

 Schema

 Design

 My Scripts

 **Code Library**

Code Library

JSON

JavaOne - Simplified and Fast Fraud Detection using SQL Pattern Matching

This session demonstrates how, using SQL pattern matching techniques, you can speed up searching for fraudulent

Tutorial 19 8 months ago

Sessionization with MATCH_RECOGNIZE and JSON

How to use new 12c SQL pattern matching match_recognize feature for sessionization analysis based on

Tutorial 13 1.1 years ago

SQL/JSON Features in Database 12.2

This Tutorial introduces the new JSON-related extensions to SQL and PL/SQL that have been added to

Tutorial 12 1.4 years ago

Area

All

Category

All

Types

☐ All

☒ **Tutorials**

☐ Scripts

Sort By

☐ Date Added

☒ **Executions**

☐ Name

☐ Likes

livesql.oracle.com

Learn more

AskTom Office hours on JSON

Live SQL

Twitter: bch_t

<https://www.oracle.com/database/technologies/appdev/json.html>

<https://docs.oracle.com/en/database/oracle/simple-oracle-document-access/index.html>

<https://github.com/oracle/soda-for-java>

<https://github.com/oracle/node-oracledb>

https://github.com/oracle/python-cx_Oracle

ORACLE®

Dealing with JSON data in Oracle: DOs and DON'Ts

Beda Hammerschmidt
beda.hammerschmidt@oracle.com
Twitter: **bch_t**
Consulting Member of Technical Staff
Oracle, Database Group, JSON support
May 15th 2019, CERN, Geneva



**Live for
the Code**

ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved. | Confidential – Oracle Internal/Restricted/Highly Restricted