DBAplus

数据库技术沙龙-武汉站

ORACLE 12C JSON实战

袁伟翔

个人简介

袁伟翔 （Buddy Yuan）
微博: Buddy_Yuan
邮箱:yuanweixiang@shsnc.com

2005年武汉工程大学计算机系毕业。从事数据库类开发。码农一个
2007年开始转型当DBA。
2012年加入上海新炬。任专家架构师。
2013年12月5日 首届GITC全球互联网技术大会 分享<从故障诊断到故障管理>。
2014年3月 ITPUB 2014全国巡回DBA技术沙龙上海站 分享<从故障诊断到故障管理>
2014年5月 ITPUB 2014全国巡回DBA技术沙龙成都站 分享<从故障诊断到故障管理>
2014年6月 ITPUB 2014全国巡回DBA技术沙龙武汉站 分享<从故障诊断到故障管理>
2017年7月 DBA PLUS分享《脑洞大开，使用Perf深入研究Oracle内部》

擅长:
故障处理。内核追踪。Oracle升级

DBAplus

一、Json简介

二、Json在12c中的运用

三、Json和索引

什么是JSON,JSON的历史

JSON是JavaScript Object Notation的缩写，它是一种数据交换格式。

在JSON出现之前，大家一直用XML来传递数据。因为XML是一种纯文本格式，所以它适合在网络上交换数据。XML本身不算复杂，但是，加上DTD、XSD、XPath、XSLT等一大堆复杂的规范以后，就变得比较复杂了

2002年的一天，道格拉斯·克罗克福特（Douglas Crockford）发明了JSON这种超轻量级的数据交换格式。他设计的JSON实际上是JavaScript的一个子集。

由于JSON非常简单，很快就风靡Web世界，并且成为ECMA标准。

JSON’s structure

基于2个结构

object: {}

array: []

对象由键值对构成

值可以是下列任意类型:

string: “test”

number: 100

Boolean: true or false

structure: object or array

no value: null

JSON’s structure

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700

```
1  {
2    "DEPARTMENT_ID": 10,
3    "DEPARTMENT_NAME": "Administration",
4    "MANAGER_ID": 200,
5    "LOCATION_ID": 1700
6  }
```

JSON's structure

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	114	1700

```
1  [
2    {
3      "id": 10,
4      "name": "Administration",
5      "managerId": 200,
6      "locationId": 1700
7    },
8    {
9      "id": 20,
10     "name": "Marketing",
11     "managerId": 201,
12     "locationId": 1800
13   },
14   {
15     "id": 30,
16     "name": "Purchasing",
17     "managerId": 114,
18     "locationId": 1700
19   }
20 ]
```

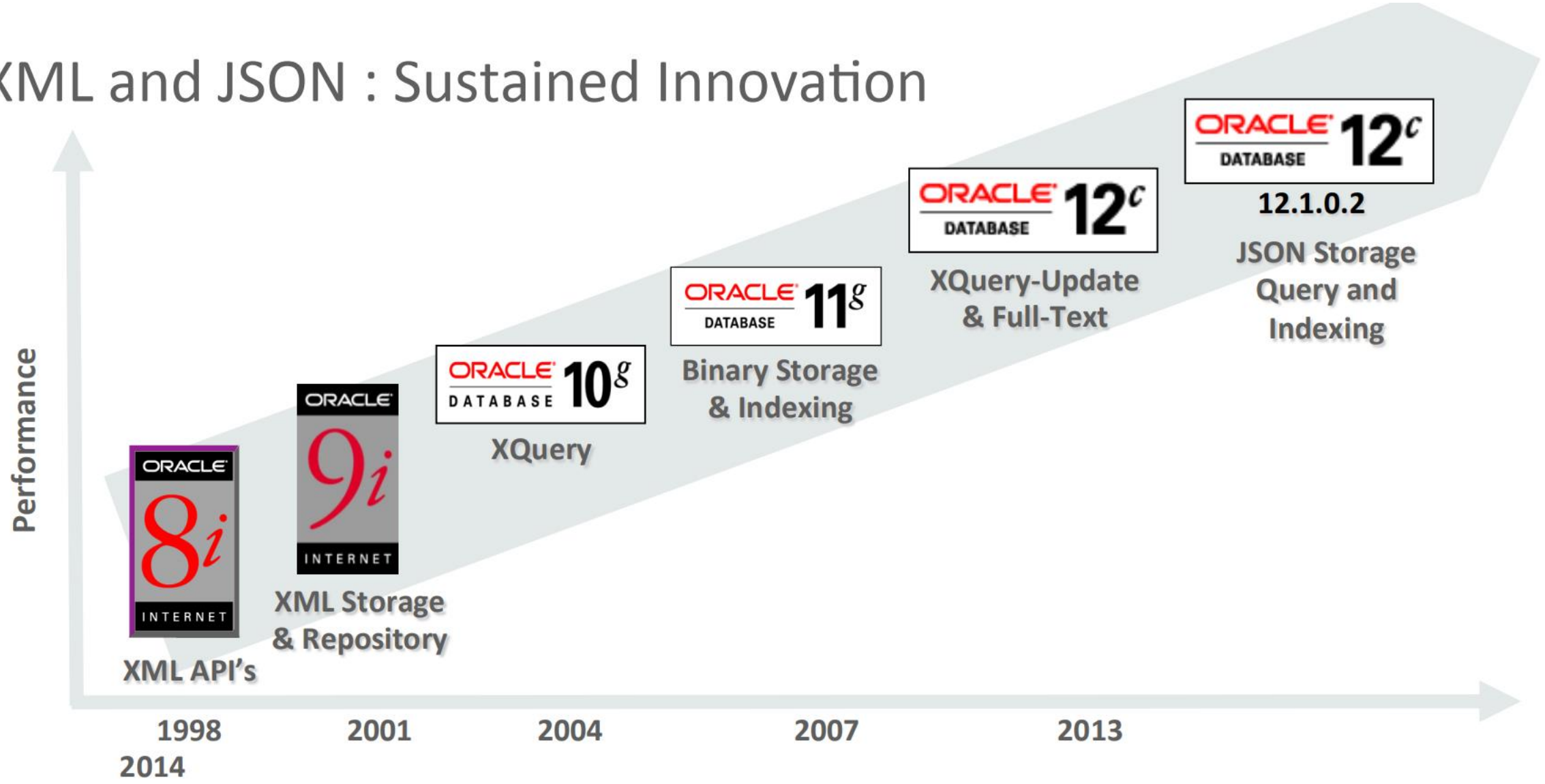
JSON’s structure

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	20	Marketing	201	1800

	EMPLOYEE_ID	DEPARTMENT_ID	NAME	SALARY	HIRE_DATE
1	201	20	Michael Hartstein	13000	17-FEB-04
2	202	20	Pat Fay	6000	17-AUG-05

```
{
  "id": 20,
  "name": "Marketing",
  "managerId": 201,
  "locationId": 1800,
  "employees": [
    {
      "id": 201,
      "name": "Michael Hartstein",
      "salary": 13000,
      "hireDate": "2004-02-17T00:00:00Z"
    },
    {
      "id": 202,
      "name": "Pat Fay",
      "salary": 6000,
      "hireDate": "2005-08-17T00:00:00Z"
    }
  ]
}
```

XML and JSON : Sustained Innovation



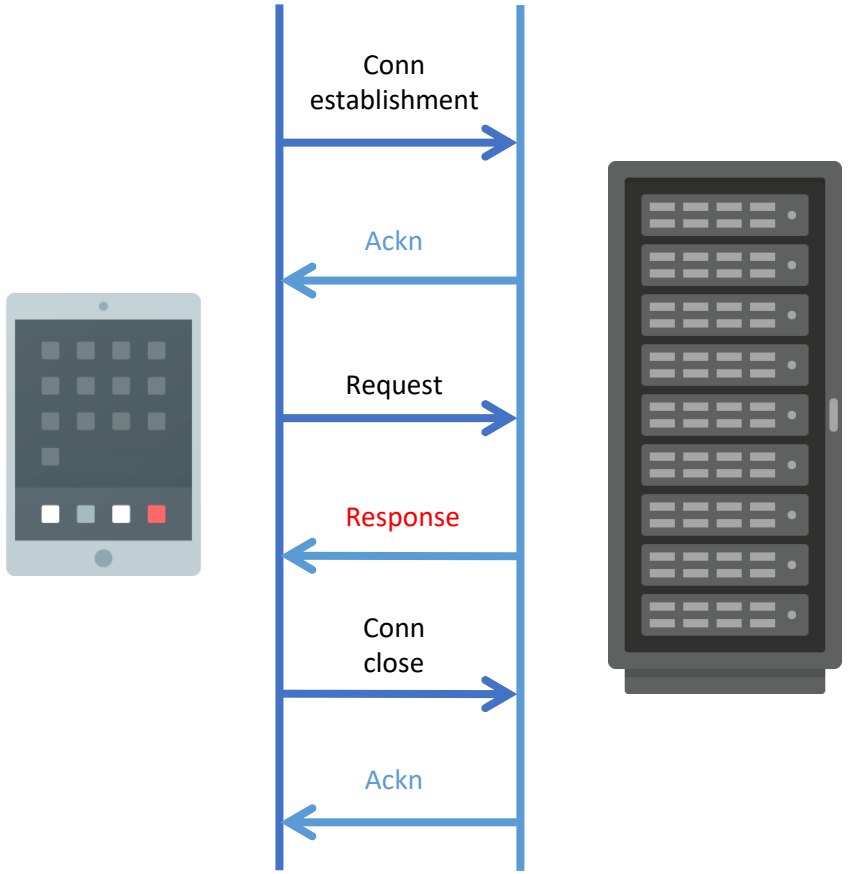
JSON和XML的区别

JSON vs. XML (Data Exchange Formats)

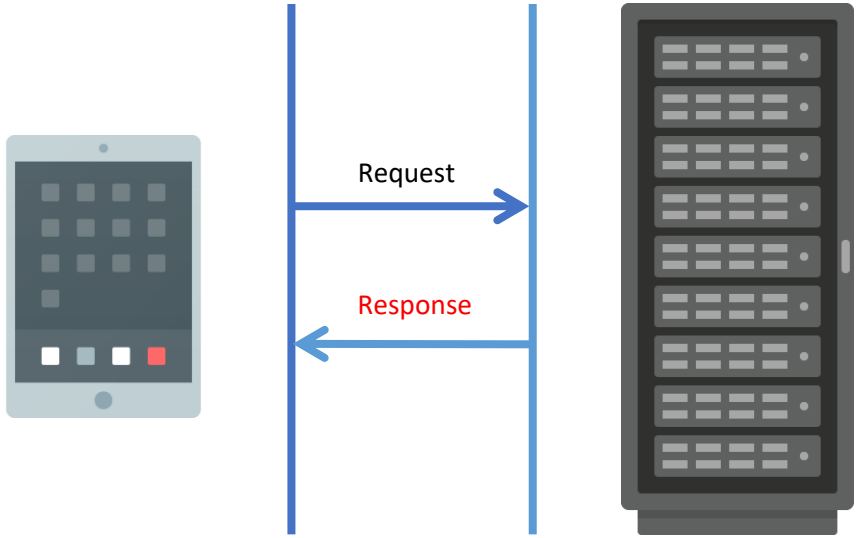
Criteria	JSON	XML
AJAX	The X in AJAX	
	JavaScript baby	Needs more libraries
Namespaces	No	Yes
Input validation	No	Yes
Transmission time	less	more
Verbosity	less	more
Web services	REST	SOAP
	REST is more popular in non-critical applications	SOAP is more reliable is business applications

XML vs JSON

Conventional communication



RESTful communication



为什么考虑文档存储？

Relational

Document Data Structure

```
{   customer_id : 1,
    first_name : "Mark",
    last_name : "Smith",
    city : "San Francisco",
    location : [40.74, -73.97],
    image : <binary>,
    phones: [    {
          number : "1-212-777-1212",
dnc : true,
          type : "home"
    },
    {
          number : "1-212-777-1213",
          type : "cell"
    }]
}
```

15

- 读写效率高。

正常情况，在关系数据库中，我们需要读取两个表关联，那么就需要物理访问两个表，而这两个表有可能不在同一块磁盘上，或者这两个表在硬盘上不同位置。需要磁头去花时间定位。

- 可扩展能力强。

关系数据库如果数据量太大之后。做水平拆库会很麻烦，因为涉及到关联关系。而文档数据库因为没有关联关系，扩展非常简单。

- 非预定义模式

可以支持动态增加字段，能够动态满足很多新的需求。

- 模型和对象模型接近

在我们日常编程的过程中，一般会使用面向对象。而对象模型和文档模型是接近的，这样就不需要做对象关系映射。

一个很牛的案例？

Twitter data (from the Twitter API)


```
{
  + retweeted_status: {…},
    contributors: null,
    text: "RT @iAdvise_live: #cou
    geo: null,
    retweeted: false,
    in_reply_to_screen_name: null,
    truncated: false,
    lang: "nl",
  + entities: {…},
    in_reply_to_status_id_str: nu
    id: 578903819884585000,
    source: "<a href="http://twit
    in_reply_to_user_id_str: null,
    favorited: false,
    in_reply_to_status_id: null,
    retweet_count: 2,
    created_at: "Fri Mar 20 13:00
    in_reply_to_user_id: null,
    favorite_count: 0,
    id_str: "578903819884584960",
    place: null,
  - user: {
      location: "",
      default_profile: false,
      profile_background_tile: true,
      statuses_count: 266,
      lang: "en",
      profile_link_color: "009999",
      profile_banner_url: "https://pbs.twimg.com/profi
      id: 1568150293,
      following: false,
      protected: false,
      profile_location: null,
      favourites_count: 62,
      profile_text_color: "333333",
      description: "Vader van Justus|Man van Jet|Comm
      verified: false,
      contributors_enabled: false,
      profile_sidebar_border_color: "EEEEEE",
      name: "Jonathan van Vianen",
      profile_background_color: "131516",
      created_at: "Thu Jul 04 12:44:14 +0000 2013",
      is_translation_enabled: false,
      default_profile_image: false,
      followers_count: 187,
      profile_image_url_https: "https://pbs.twimg.com/
      geo_enabled: false,
      profile_background_image_url: "http://abs.twimg.
      profile_background_image_url_https: "https://ab
      follow_request_sent: false,
    + entities: {…},
      url: "http://t.co/DNhI2jUKjh",
      utc_offset: 3600,
      time_zone: "Amsterdam",
      profile_sidebar_border_color: "EEEEEE",
      name: "Jonathan van Vianen",
      profile_background_color: "131516",
      created_at: "Thu Jul 04 12:44:14 +0000 2013",
      is_translation_enabled: false,
      default_profile_image: false,
      followers_count: 187,
      profile_image_url_https: "https://pbs.twimg.com/profile_images/5
      geo_enabled: false,
      profile_background_image_url: "http://abs.twimg.com/images/theme
      profile_background_image_url_https: "https://abs.twimg.com/image
      follow_request_sent: false,
    + entities: {…},
      url: "http://t.co/DNhI2jUKjh",
      utc_offset: 3600,
      time_zone: "Amsterdam",
      notifications: false,
      profile_use_background_image: true,
      friends_count: 493,
      profile_sidebar_fill_color: "EFEFEF",
      screen_name: "jvanvianen78",
      id_str: "1568150293",
      profile_image_url: "http://pbs.twimg.com/profile_images/5256952
      listed_count: 13,
      is_translator: false
    },
    coordinates: null,
  + metadata: {…}
}
```


Twitter API提供了Twitter部分数据

DBAplus

一个很牛的案例？

Tweet API提供的JOSN如下：

1 Tweet = 140 Characters
1 JSON = ~4900 Characters on average

你肯定会问为什么？140个字符怎么变成平均4900个字符？

一个很牛的案例？

一条完整的Twitter：

包含：完整的用户信息

回复留言

转发信息

主题标签

等……

一个很牛的案例？

一条完整的Twitter：

包含：完整的用户信息

回复留言

转发信息

主题标签

等……

一个很牛的案例？

1.下载“twitter4j” 这个Java Client library

2. 加载“twitter4j” 到Oracle Database中.

3.创建Java存储过程访问Twitte，将这个映射成PL/SQL函数

4.准备表、视图，包。

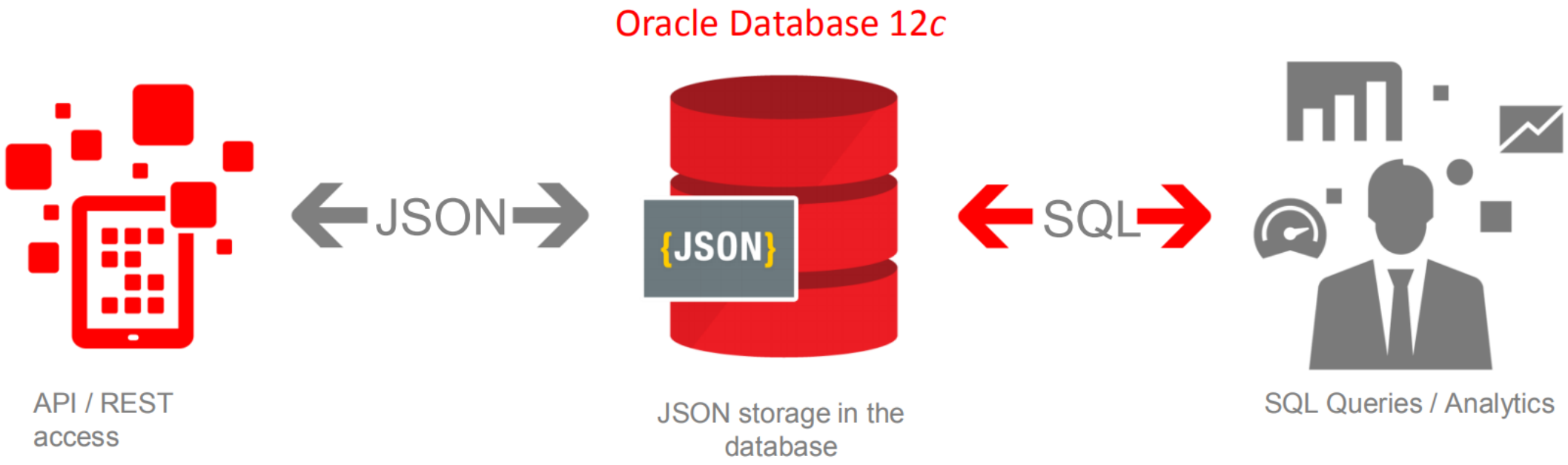
一个很牛的案例？

```
SQL> desc tweet_pkg
FUNCTION GET_GRANTS_SCRIPT RETURNS VARCHAR2
FUNCTION GET_TWEETS_JSON RETURNS TWEET_JSON_CT
 Argument Name                  Typ                     In/Out Defaultwert?
 ------------------------------ ----------------------- ------ --------
 P_SEARCH                       VARCHAR2                IN
 P_ID                           NUMBER                  IN
PROCEDURE STORE_TWEETS
 Argument Name                  Typ                     In/Out Defaultwert?
 ------------------------------ ----------------------- ------ --------
 P_CATID                        NUMBER                  IN
PROCEDURE STORE_TWEETS

SQL> select * from table(tweet_pkg.get_tweets_json('@ogh_nl #apexworld', -1)) where rownum <= 10;

COLUMN_VALUE
--------------------------------------------------------------------------------
{"retweeted":false,"in_reply_to_screen_name":null,"possibly_sensitive":false,"tr
{"retweeted":false,"in_reply_to_screen_name":null,"possibly_sensitive":false,"tr
{"retweeted":false,"in_reply_to_screen_name":null,"possibly_sensitive":false,"tr
:
```

一个很牛的案例？

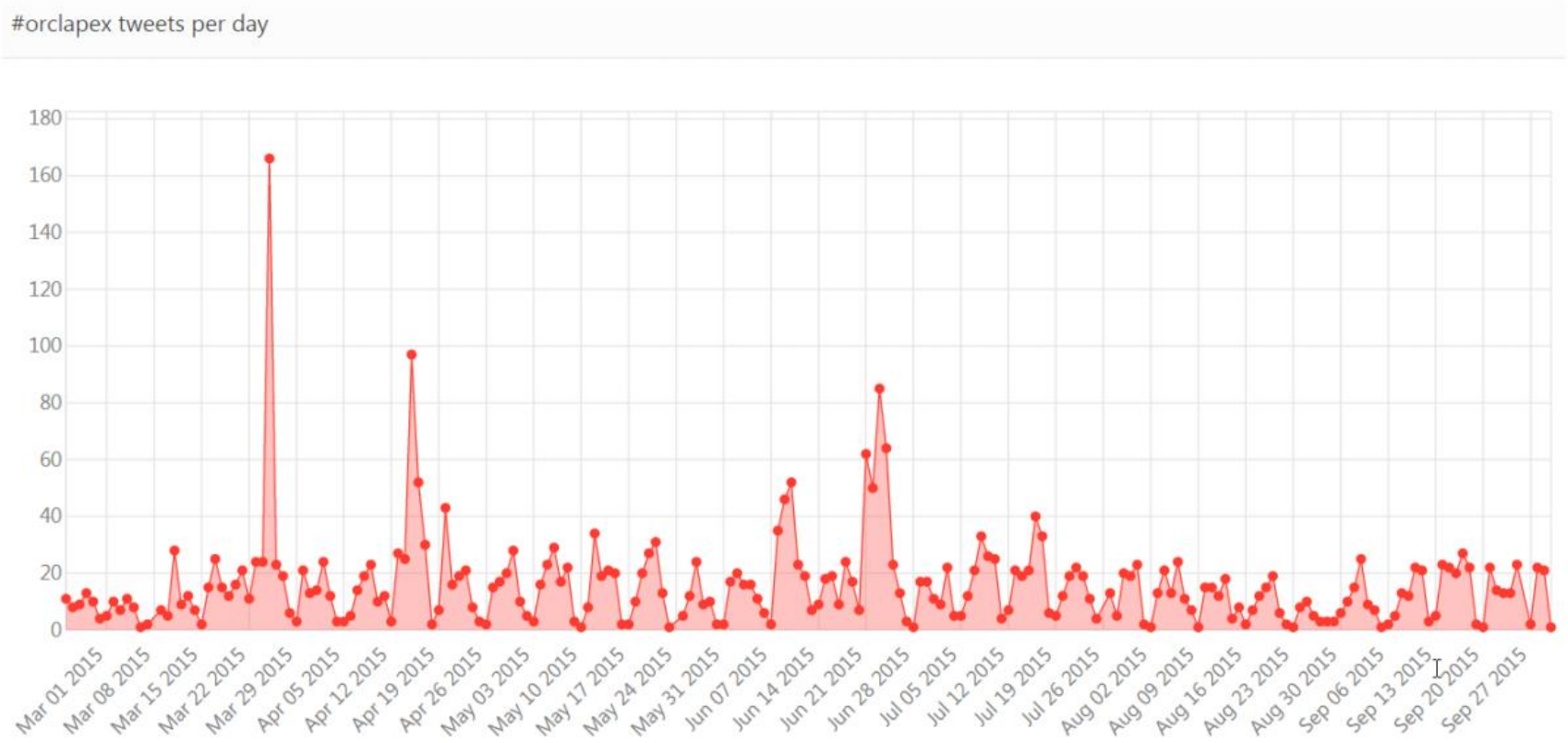
```
SQL> select tweet from apextweets where rownum <= 3;

TWEET
--------------------------------------------------------------------------------
{"retweeted_status":{"contributors":null,"text":"#countdown naar #Apexworld @OGh
_nl #iadvise_live @Yvke1983 geeft met #Robeco een client case over #twitterboots
trap #formsmigratie #orclapex","geo":null,"retweeted":false,"in_reply_to_screen_
name":null,"truncated":false,"lang":"nl","entities":{"symbols":[],"urls":[],"has
htags":[{"text":"countdown","indices":[0

{"retweeted_status":{"contributors":null,"text":"#countdown naar #Apexworld @OGh
_nl #iadvise_live @Yvke1983 geeft met #Robeco een client case over #twitterboots
trap #formsmigratie #orclapex","geo":null,"retweeted":false,"in_reply_to_screen_
name":null,"truncated":false,"lang":"nl","entities":{"symbols":[],"urls":[],"has
htags":[{"text":"countdown","indices":[0

{"contributors":null,"text":"#countdown naar #Apexworld @OGh_nl #iadvise_live @Y
vke1983 geeft met #Robeco een client case over #twitterbootstrap #formsmigratie
#orclapex","geo":null,"retweeted":false,"in_reply_to_screen_name":null,"truncate
d":false,"lang":"nl","entities":{"symbols":[],"urls":[],"hashtags":[{"text":"cou
ntdown","indices":[0,10]},{"text":"Apexw
```

一个很牛的案例？


Oracle Database 12c
JSON
{JSON}
SQL
API / REST access
JSON storage in the database
SQL Queries / Analytics

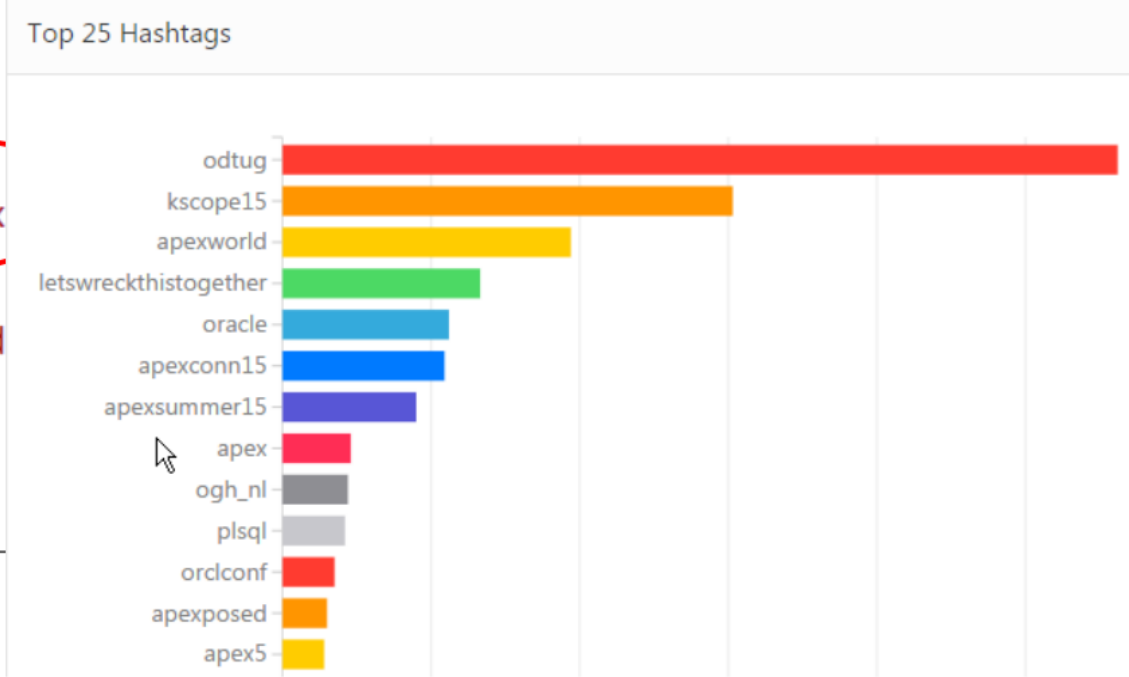
一个很牛的案例？

- JSON Attribute: **$.created_at**



一个很牛的案例？

```
select * from (
  select
    lower(hashtag) as hashtag,
    count(*)       as cnt
  from apextweets, json_table(
    tweet,
    '$.entities.hashtags[*]'
    columns (
      hashtag     varchar2(30)  path '$.tex
    )
  )
  where not json_exists(tweet, '$.retweeted
  group by lower(hashtag)
  order by 2 desc
)
where rownum <= 25
```



DBAplus

一、Json简介

二、Json在12c中的运用

三、Json和索引

表中包含JSON文档

```
SQL> CREATE TABLE json_documents (
    id    RAW(16) NOT NULL,
   data  CLOB,
   CONSTRAINT json_documents_pk PRIMARY KEY (id),
   CONSTRAINT json_documents_json_chk   CHECK (data IS JSON)
  );
```

表中包含Json文档。没有指定特殊的数据类型，可以是Varchar2, CLOB, BLOB

MongoDB数据库使用的是BSON类型(二进制的JSON)

IS JSON约束确保只有合法的JSON文档可以插入

插入JSON文档

```
INSERT INTO json_documents (id, data)
VALUES (SYS_GUID(),
        '{
          "FirstName"      : "John",
          "LastName"       : "Doe",
          "Job"            : "Clerk",
          "Address"        : {
                              "Street"   : "99 My Street",
                              "City"     : "My City",
                              "Country"  : "UK",
                              "Postcode" : "A12 34B"
                             },
          "ContactDetails" : {
                              "Email"    : "john.doe@example.com",
                              "Phone"    : "44 123 123456",
                              "Twitter"  : "@johndoe"
                             },
          "DateOfBirth"    : "01-JAN-1980",
          "Active"         : true
         }');

INSERT INTO json_documents (id, data)
VALUES (SYS_GUID(),
        '{
          "FirstName"      : "Jayne",
          "LastName"       : "Doe",
          "Job"            : "Manager",
          "Address"        : {
                              "Street"   : "100 My Street",
                              "City"     : "My City",
                              "Country"  : "UK",
                              "Postcode" : "A12 34B"
                             },
          "ContactDetails" : {
                              "Email"    : "jayne.doe@example.com",
                              "Phone"    : ""
                             },
          "DateOfBirth"    : "01-JAN-1982",
          "Active"         : false
         }');
```

查询JSON文档

```
SELECT a.data.FirstName,a.data.LastName,a.data.Address.Postcode AS Postcode,
 a.data.ContactDetails.Email AS Email
 FROM   json_documents a ORDER BY a.data.FirstName,a.data.LastName;

FIRSTNAME       LASTNAME        POSTCODE   EMAIL
--------------- --------------- ---------- -------------------------
Jayne           Doe             A12 34B    jayne.doe@example.com
John            Doe             A12 34B    john.doe@example.com
```

如果查询的列具备一个IS JSON的check约束。则可以使用点来直接从SQL中查询文档中的各个元素。

```
SELECT a.data.ContactDetails.Email FROM json_documents a;
CONTACTDETAILS
--------------------------------------------------------------------------------
john.doe@example.com
jayne.doe@example.com
```

这里需要注意一点，当我们用点向下钻入下一个元素读取数据时，查询的列名仍然为顶级元素

查询JSON文档

```
SQL> SELECT a.data.ContactDetails FROM  json_documents a;

CONTACTDETAILS
--------------------------------------------------------------------------------
{"Email":"john.doe@example.com","Phone":"44 123 123456","Twitter":"@johndoe"}
{"Email":"jayne.doe@example.com","Phone":""}
```

如果查询的元素是一个非标量值，结果会作为JSON片段返回。

JSON操作之IS JSON

IS JSON可以帮助我们验证是否包含JSON数据，使用方法就和IS NULL一样。

```
CREATE TABLE json_documents_no_constraint (
  id    RAW(16) NOT NULL,
  data  CLOB,
  CONSTRAINT json_documents_nocon_pk PRIMARY KEY (id)
);

INSERT INTO json_documents_no_constraint VALUES (SYS_GUID(), '{"FirstName" : "John"}');
INSERT INTO json_documents_no_constraint VALUES (SYS_GUID(), 'John');

SQL> select * from json_documents_no_constraint a WHERE  a.data IS JSON;

ID                                DATA
--------------------------------- --------------------------------------------------------------------------------
5E7EFAF9CB3972EFE0537838A8C093FD {"FirstName" : "John"}
```

当然你也可以使用is not josn来查找不是json文档的数据。

JSON操作函数之JSON_EXISTS

```
SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a
WHERE  JSON_EXISTS(a.data.ContactDetails, '$.Phone' FALSE ON ERROR)

FIRSTNAME       LASTNAME        EMAIL
--------------- --------------- -------------------------
John            Doe             john.doe@example.com
Jayne           Doe             jayne.doe@example.com

SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a

WHERE  NOT JSON_EXISTS(a.data.ContactDetails, '$.Twitter' FALSE ON ERROR)
FIRSTNAME       LASTNAME        EMAIL
--------------- --------------- -------------------------
Jayne           Doe             jayne.doe@example.com
```

JSON_EXISTS可以帮助我们区分空元素和缺少元素，上述查询帮助我们查找Phone元素存在的，和Twitter元素不存在的。当Phone是空元素时也能被查出来。

JSON操作函数之JSON_EXISTS

```
SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a
WHERE  JSON_EXISTS(a.data.ContactDetails, '$.Phone' FALSE ON ERROR)

FIRSTNAME       LASTNAME        EMAIL
--------------- --------------- -------------------------
John            Doe             john.doe@example.com
Jayne           Doe             jayne.doe@example.com

SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a

WHERE  NOT JSON_EXISTS(a.data.ContactDetails, '$.Twitter' FALSE ON ERROR)
FIRSTNAME       LASTNAME        EMAIL
--------------- --------------- -------------------------
Jayne           Doe             jayne.doe@example.com
```

默认处理方式是FALSE ON ERROR
出现错误返回False

JSON_EXISTS可以帮助我们区分空元素和缺少元素，上述查询帮助我们查找Phone元素存在的，和Twitter元素不存在的。当Phone是空元素时也能被查出来。

JSON操作函数之JSON_VALUE

```
SELECT JSON_VALUE(a.data, '$.FirstName') AS first_name,
       JSON_VALUE(a.data, '$.LastName') AS last_name
FROM   json_documents a ORDER BY 1, 2;

FIRST_NAME      LAST_NAME
--------------- ---------------
Jayne           Doe
John            Doe

SELECT JSON_VALUE(a.data, '$.ContactDetails') AS contact_details FROM json_documents a ORDER BY 1;

-----------这里返回了空，因为路径ContactDetails为非标量值，默认的处理方式就是NULL ON ERROR

SELECT JSON_VALUE(a.data, '$.ContactDetails' ERROR ON ERROR) AS contact_details FROM json_documents a ORDER BY 1;
ERROR at line 2:
ORA-40456: JSON_VALUE evaluated to non-scalar value

----------------------这里如果写ERROR ON ERROR，就会真实的返回一个错误，告诉你不是一个标量值.
```

JSON_VALUE函数根据指定的JSON路径从文档中返回一个元素，它只会返回一个标量值

JSON操作函数之JSON_VALUE

```
SELECT a.data.FirstName, a.data.LastName,
JSON_VALUE(a.data, '$.Active') AS Active,
JSON_VALUE(a.data, '$.Active' RETURNING NUMBER) AS ActiveNum
FROM json_documents a ORDER BY a.data.FirstName, a.data.LastName;

FIRSTNAME       LASTNAME        ACTIVE      ACTIVENUM
--------------- --------------- ---------- ----------
Jayne           Doe             false               0
John            Doe             true                1
```

JSON_VALUE可以把True/False转换成数字1和0

JSON操作函数之JSON_QUERY

```
SELECT a.data.FirstName,
a.data.LastName,
JSON_QUERY(a.data, '$.ContactDetails'  WITH WRAPPER) AS contact_details
FROM   json_documents a
ORDER BY a.data.FirstName,
a.data.Last_name;

FIRSTNAME       LASTNAME        CONTACT_DETAILS
--------------- --------------- ----------------------------------------------------------------------------------------------------
Jayne           Doe             [{"Email":"jayne.doe@example.com","Phone":""}]
John            Doe             [{"Email":"john.doe@example.com","Phone":"44 123 123456","Twitter":"@johndoe"}]
```

JSON_QUERY函数返回一个表示一个或多个值的JSON片段。 在以下示例中，JSON_QUERY用于返回一个JSON片段，表示每个人的所有联系信息。 WITH WRAPPER选项用方括号括住片段。

JSON操作函数之JSON_TABLE

```
SELECT jt.first_name,jt.last_name,
       jt.job,jt.addr_street,jt.addr_city,jt.addr_country,
       jt.addr_postcode,jt.email,jt.phone,jt.twitter,
       TO_DATE(jt.dob, 'DD-MON-YYYY') AS dob,
       jt.active
FROM   json_documents,
       JSON_TABLE(data, '$'
       COLUMNS (first_name    VARCHAR2(50 CHAR) PATH '$.FirstName',
                last_name     VARCHAR2(50 CHAR) PATH '$.LastName',
                job           VARCHAR2(10 CHAR) PATH '$.Job',
                addr_street   VARCHAR2(50 CHAR) PATH '$.Address.Street',
                addr_city     VARCHAR2(50 CHAR) PATH '$.Address.City',
                addr_country  VARCHAR2(50 CHAR) PATH '$.Address.Country',
                addr_postcode VARCHAR2(50 CHAR) PATH '$.Address.Postcode',
                email         VARCHAR2(100 CHAR) PATH '$.ContactDetails.Email',
                phone         VARCHAR2(50 CHAR) PATH '$.ContactDetails.Phone',
                twitter       VARCHAR2(50 CHAR) PATH '$.ContactDetails.Twitter',
                dob           VARCHAR2(11 CHAR) PATH '$.DateOfBirth',
                active        VARCHAR2(5 CHAR) PATH  '$.Active')) jt;
```

JSON_TABLE函数结合了JSON_VALUE，JSON_EXISTS和JSON_QUERY的所有功能。 比使用其他单独的JSON函数的语法稍微复杂一些，但使用单个JSON_TABLE比将多个调用与其他单个函数组合成单个查询更有效。 COLUMNS子句定义了每列的数据如何被识别和呈现。

JSON操作函数之JSON_TABLE

```
SELECT jt.first_name,
       jt.last_name,
       jt.contact_details,
       jt.Street,
       jt.City
FROM   json_documents,
       JSON_TABLE(data, '$'
         COLUMNS (first_name    VARCHAR2(50 CHAR) PATH '$.FirstName',
                  last_name     VARCHAR2(50 CHAR) PATH '$.LastName',
                  contact_details  VARCHAR2(4000 CHAR) FORMAT JSON WITH WRAPPER PATH '$.ContactDetails',
                  NESTED PATH '$.Address'
                   columns(
                   Street VARCHAR2(50 CHAR)     PATH '$.Street',
                   City   VARCHAR2(50 CHAR)     PATH '$.City')
                  )) jt;

FIRST_NAME LAST_NAME  CONTACT_DETAILS                                                                  STREET           CITY
---------- ---------- ------------------------------------------------------------------------------- ---------------- ----------
John       Doe        [{"Email":"john.doe@example.com","Phone":"44 123 123456","Twitter":"@johndoe"}]  99 My Street     My City
Jayne      Doe        [{"Email":"jayne.doe@example.com","Phone":""}]                                  100 My Street    My City
```

遍历JSON文档有多种选择，包括使用NESTED子句处理数组，并控制数据的呈现方式。 在以下示例中，联系人详细信息以JSON格式显示。在Oracle 12.2中，对JSON_EXISTS，JSON_VALUE和JSON_QUERY的多次调用可能会被重写为更少的JSON_TABLE调用，以提高性能。

如何确认列包含JSON文档

```
COLUMN table_name FORMAT A15
COLUMN column_name FORMAT A15

SELECT table_name,
       column_name,
       format,
       data_type
FROM   user_json_columns;

TABLE_NAME      COLUMN_NAME     FORMAT    DATA_TYPE
--------------- --------------- --------- -------------
JSON_DOCUMENTS  DATA            TEXT      CLOB
```

[USER | ALL | DBA] _JSON_COLUMNS视图可用于识别包含JSON数据的表和列。

能用.把数据选出来为什么还需要JSON_TABLE/JSON_QUERY等函数

```
ALTER SESSION SET EVENTS '10053 trace name context forever';

SELECT a.data.FirstName,
       a.data.LastName
FROM   json_documents a;

ALTER SESSION SET EVENTS '10053 trace name context off';
```

让我们做10053 Trace看看会发生什么?

```
Final query after transformations:******* UNPARSED QUERY IS *******
SELECT "P"."C_02$" "FIRSTNAME","P"."C_01$" "LASTNAME" FROM "A1"."JSON_DOCUMENTS" "A",JSON_TABLE( "A"."DATA", '$' COLUMNS( "C_01$" VARCHAR2(
4000) PATH '$.LastName' ASIS NULL ON ERROR , "C_02$" VARCHAR2(4000) PATH '$.FirstName' ASIS NULL ON ERROR ) )  "P"
kkoqbc: optimizing query block SEL$2919ABAD (#0)
```

Oracle自动通过查询转换把.转换成了JSON_TABLE函数，并且使用了NULL ON ERROR处理方式

DBAplus

一、Json简介

二、Json在12c中的运用

三、Json和索引

JSON文档上索引需要注意的点

1.开发人员使用文档存储。而添加索引将影响DML的性能，DML操作必须维护索引。所以必须权衡DML性能对索引开销与改进的查询性能之间的关系。

2.大多数JSON索引是基于函数的索引，这也意味着维护的开销要高于常规的B*树索引。

3.如果JSON文档非常大，那么索引的开销可能更大。

4.Oracle优化器对JSON索引非常挑剔，参数的轻微变化可以使特定查询语句下，索引不能使用。

5.对于位图索引和全文索引，同样的规则也适用。

Function-Based Indexes

```
CREATE INDEX json_docs_email_idx
  ON json_documents (JSON_VALUE(data, '$.ContactDetails.Email' RETURNING VARCHAR2 ERROR ON ERROR));

SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a
WHERE  JSON_VALUE(data, '$.ContactDetails.Email' RETURNING VARCHAR2 ERROR ON ERROR) = 'john.doe@example.com';

Execution Plan
----------------------------------------------------------
Plan hash value: 3327373981

-----------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                      |   163 |   244K|    31   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                        |                      |   163 |   244K|    31   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| JSON_DOCUMENTS       |     1 |  1527 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                  | JSON_DOCS_EMAIL_IDX  |     1 |       |     1   (0)| 00:00:01 |
|   4 |   JSONTABLE EVALUATION               |                      |       |       |            |          |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access(JSON_VALUE("DATA" FORMAT JSON , '$.ContactDetails.Email' RETURNING VARCHAR2(4000)
              ERROR ON ERROR)='john.doe@example.com')
```

Function-Based Indexes

```
SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a
WHERE  a.data.ContactDetails.Email = 'john.doe@example.com';

Execution Plan
----------------------------------------------------------
Plan hash value: 3327373981

-----------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                     |   163 |   244K|    31   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                        |                     |   163 |   244K|    31   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| JSON_DOCUMENTS      |     1 |  1527 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                  | JSON_DOCS_EMAIL_IDX |     1 |       |     1   (0)| 00:00:01 |
|   4 |   JSONTABLE EVALUATION               |                     |       |       |            |          |
-----------------------------------------------------------------------------------------------------
 3 - access(JSON_VALUE("DATA" FORMAT JSON , '$.ContactDetails.Email' RETURNING VARCHAR2(4000)
              ERROR ON ERROR)='john.doe@example.com')
```

尽管我们使用点表示法，谓词实际上是作为一个json_value能够使用函数索引。这里需要注意一点，默认使用.创建索引，它会使用JSON_QUERY函数来进行创建，而如果已经指定了JSON_VALUE函数来创建索引，则使用.的时候可以用查询转换来执行对JSON_VALUE函数索引的调用。

使用.创建索引呢?

```
drop index json_docs_email_idx;

CREATE INDEX json_docs_email_idx ON json_documents a (a.data.ContactDetails.Email);

SELECT a.data.FirstName, a.data.LastName, a.data.ContactDetails.Email AS Email FROM json_documents a WHERE a.data.ContactDetails.Email
= 'john.doe@example.com';

Execution Plan
----------------------------------------------------------
Plan hash value: 3327373981

-----------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |                      |   163 |   243K|    31   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                        |                      |   163 |   243K|    31   (0)| 00:00:01 |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| JSON_DOCUMENTS       |     1 |  1527 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                  | JSON_DOCS_EMAIL_IDX  |     1 |       |     1   (0)| 00:00:01 |
|   4 |   JSONTABLE EVALUATION               |                      |       |       |            |          |
-----------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access(JSON_QUERY("DATA" FORMAT JSON , '$.ContactDetails.Email' RETURNING VARCHAR2(4000)
              ASIS  WITHOUT ARRAY WRAPPER NULL ON ERROR)='john.doe@example.com')
```

可以看到，我们用.创了索引，而查询语句也使用了点，这里走了.默认使用的JSON_QUERY函数索引

用.创建的索引，查询条件使用JSON_VALUE会走索引吗?

```
SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a
WHERE  JSON_VALUE(data, '$.ContactDetails.Email' RETURNING VARCHAR2 ERROR ON ERROR) = 'john.doe@example.com';

Execution Plan
----------------------------------------------------------
Plan hash value: 2285650440

-----------------------------------------------------------------------------------------
| Id  | Operation                | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                |   163 |   244K|    59   (0)| 00:00:01 |
|   1 |  NESTED LOOPS            |                |   163 |   244K|    59   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL      | JSON_DOCUMENTS |     2 |  3054 |     3   (0)| 00:00:01 |
|*  3 |   JSONTABLE EVALUATION   |                |       |       |            |          |
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("P"."C_01$"='john.doe@example.com')
```

这里可以看到用.创建的默认是创建的JSON_QUERY函数索引，而我们使用JSON_VALUE查询并没有找到合适的基于函数的索引。因此没有使用索引

用.创建的索引，使用JSON_QUERY必须完全匹配?

```
SELECT a.data.FirstName,
       a.data.LastName,
       a.data.ContactDetails.Email AS Email
FROM   json_documents a
WHERE  JSON_QUERY("DATA" FORMAT JSON , '$.ContactDetails.Email') = 'john.doe@example.com';

Execution Plan
----------------------------------------------------------
Plan hash value: 2285650440

-----------------------------------------------------------------------------------------
| Id  | Operation                | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                |   163 |   244K|    59   (0)| 00:00:01 |
|   1 |  NESTED LOOPS            |                |   163 |   244K|    59   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL      | JSON_DOCUMENTS |     2 |  3054 |     3   (0)| 00:00:01 |
|*  3 |   JSONTABLE EVALUATION   |                |       |       |            |          |
-----------------------------------------------------------------------------------------
```

这里可以看到用.创建的JSON_QUERY函数索引我们任然使用不上。这是因为我们并没有完全匹配函数索引的定义，少了ASIS关键字。

得写成这样才会使用上索引

```
JSON_QUERY("DATA" FORMAT JSON , '$.ContactDetails.Email' RETURNING VARCHAR2(4000)
                ASIS  WITHOUT ARRAY WRAPPER NULL ON ERROR)='john.doe@example.com')
```

解决混乱的办法?

可以看到JSON创建的索引需要特别注意:一个比较简单的方法就是风格统一。标准化所有SQL语句使用JSON文档的方法，以确保可以使用上创建的索引。

第二个方法就是创建关于JSON数据的视图，查询视图而不是直接使用JSON，将会强制这种标准化，但会丧失灵活性。

复合B Tree索引?

```
ALTER TABLE json_documents ADD (first_name VARCHAR2(50) GENERATED ALWAYS AS (JSON_VALUE(data, '$.FirstName' RETURNING VARCHAR2(50))));
ALTER TABLE json_documents ADD (last_name VARCHAR2(50) GENERATED ALWAYS AS (JSON_VALUE(data, '$.LastName' RETURNING VARCHAR2(50))));

CREATE INDEX json_docs_name_idx ON json_documents (first_name, last_name);
SELECT COUNT(*)
FROM   json_documents
WHERE  first_name = 'John'
AND    last_name  = 'Doe';

-----------------------------------------------------------------------------------------
| Id  | Operation         | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                    |     1 |    54 |     1   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE   |                    |     1 |    54 |            |          |
|*  2 |   INDEX RANGE SCAN| JSON_DOCS_NAME_IDX |     1 |    54 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("FIRST_NAME"='John' AND "LAST_NAME"='Doe')
```

可以通过定义**虚拟列**来创建组合索引，对这些列使用常规索引。在内部，这仍然是一个基于函数的索引，但是这个定义看起来简单得多，它让你可以直接查询虚拟列。

复合B Tree索引?

```
CREATE INDEX json_docs_name_idx ON json_documents (
  JSON_VALUE(data, '$.FirstName' RETURNING VARCHAR2(50)),
  JSON_VALUE(data, '$.LastName' RETURNING VARCHAR2(50))
);

SELECT COUNT(*)
FROM   json_documents
WHERE  JSON_VALUE(data, '$.FirstName' RETURNING VARCHAR2(50)) = 'John'
AND    JSON_VALUE(data, '$.LastName' RETURNING VARCHAR2(50)) = 'Doe';

-----------------------------------------------------------------------------------------
| Id  | Operation         | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |                    |     1 |    54 |     1   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE   |                    |     1 |    54 |            |          |
|*  2 |   INDEX RANGE SCAN| JSON_DOCS_NAME_IDX |     1 |    54 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access(JSON_VALUE("DATA" FORMAT JSON , '$.FirstName' RETURNING
              VARCHAR2(50) NULL ON ERROR)='John' AND JSON_VALUE("DATA" FORMAT JSON ,
              '$.LastName' RETURNING VARCHAR2(50) NULL ON ERROR)='Doe')
```

当然，你要觉得不怕麻烦，也可以直接创建复合基于函数的索引。

位图索引

```
DROP INDEX json_docs_name_idx;
CREATE BITMAP INDEX json_docs_name_idx ON json_documents (
  JSON_VALUE(data, '$.FirstName')
);

DROP INDEX json_docs_name_idx;
CREATE BITMAP INDEX json_docs_name_idx ON json_documents (
  JSON_QUERY(data, '$.FirstName')
);

DROP INDEX json_docs_name_idx;
CREATE BITMAP INDEX json_docs_name_idx ON json_documents (
  JSON_EXISTS(data, '$.FirstName')
);
```

和关系型数据库使用方式一致。

Full-Text Search (JSON Search Indexes)

```
-- 12.2 Syntax
CREATE SEARCH INDEX json_docs_search_idx ON json_documents (data) FOR JSON;

SELECT COUNT(*)
FROM   json_documents
WHERE  JSON_TEXTCONTAINS(data, '$.ContactDetails.Email', 'john.doe@example.com');

-----------------------------------------------------------------------------------------
| Id  | Operation        | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                      |     1 |  2014 |     4   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE  |                      |     1 |  2014 |            |          |
|*  2 |   DOMAIN INDEX   | JSON_DOCS_SEARCH_IDX |     1 |  2014 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("CTXSYS"."CONTAINS"("JSON_DOCUMENTS"."DATA",'{john.doe@example.com}
              INPATH(/ContactDetails/Email)')>0)
```

Json搜索索引是专门为Json类型推出的一种全文索引。

DBAplus

www.dbaplus.cn

THANK YOU！