ORACLE

Oracle 21c 开发创新

SE-HUB DM team

Xu Guang

Agenda

1 21c 开发相关特性概览

2 JSON 增强特性

3 Database 中执行 Javascript

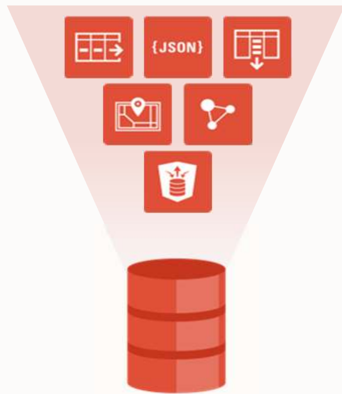
4 在 SQL 中定义 Macro

5 其他：在 JDBC 中的增强

6 Q&A

在融合数据库中更多的创新

21c



JavaScript Execution

JSON Speed & Flexibility

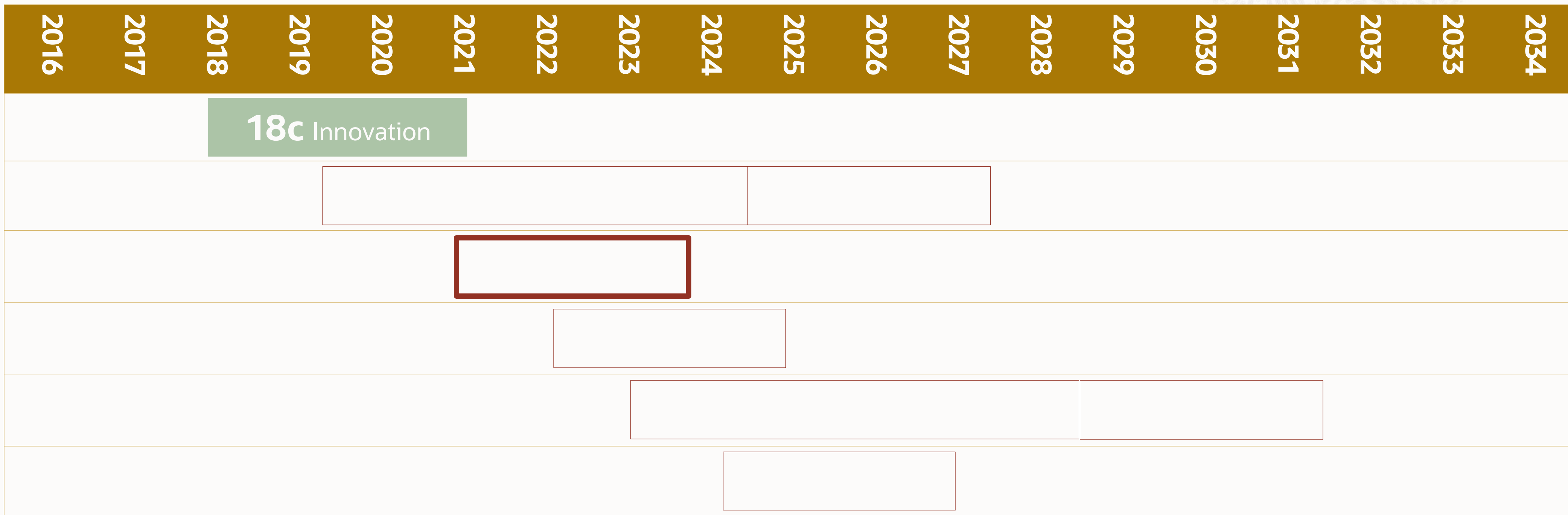
Macro in SQL

Multi-Workload

Multi-Tenant

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

Projected Database Release and Support Timeline



- Innovation Release - 2 years of Premier Support, and no Extended Support
- Long Term Release - 5 years of Premier Support, and 3 years of Extended Support

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

Enhanced set operators

PL/SQL Iterators

New stats functions

Window Clause

SQL Macros

PL/SQL Qualified Expressions

JavaScript Execution

{ JSON } Improvements

Automatic MVs

Blockchain Tables

Dynamic Sequence Cache

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

JSON 增强特性

{JSON}

JSON 数据类型

经过优化的原生JSON类型

原生 JSON 二进制 类型

- 同时支持在 SQL, PLSQL中使用
- OCI, JDBC 的原生的支持
- 基于 OSON – 经过优化的二进制表示
  - Self-contained format
  - 快速查找
  - 局部更新
- 描快速度提升**2x 以上**
- 更新速度提升 **10x 以上**

```
CREATE TABLE J_PURCHASEORDER (
    id   INTEGER PRIMARY KEY,
    po_document JSON
);
```

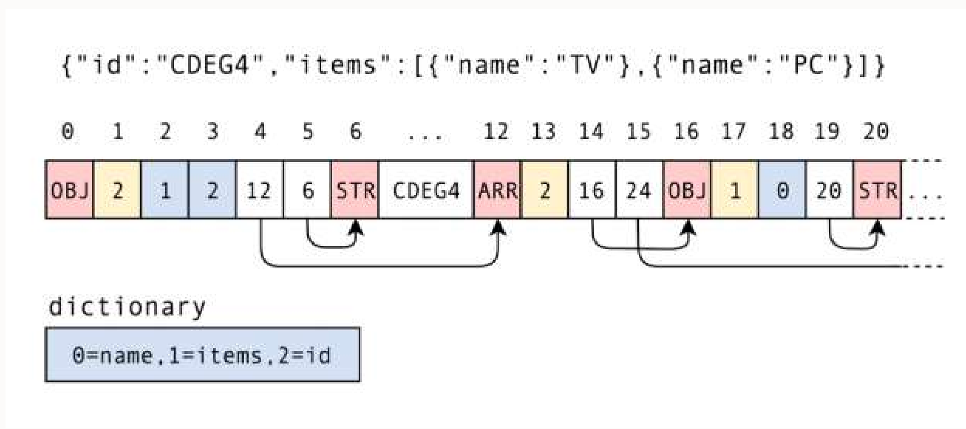
与 oracle 平台 深度集成

- 并行查询
- RAC
- Golden Gate
- …
- Leverage Advanced Security features
  - VPD, Encryption, ..
- Manage JSON together with other data
  - Operational simplicity from converged database

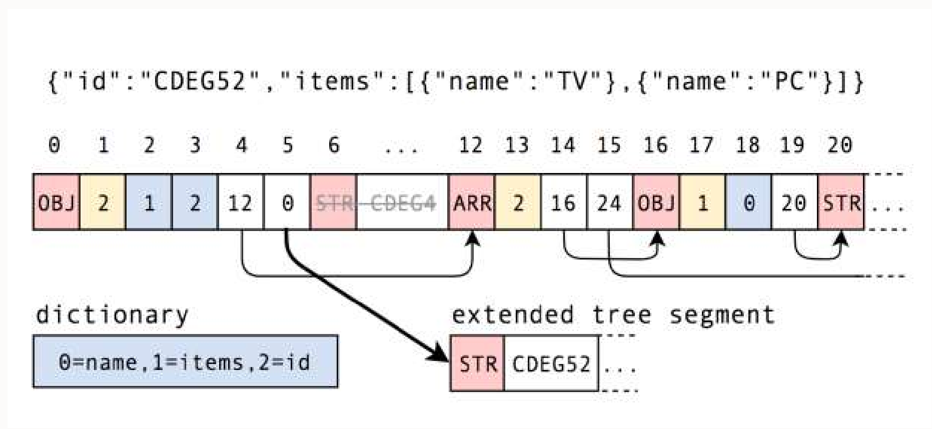
Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

JSON 数据类型

经过优化的原生JSON类型



内部存储结构



更新时的结果

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

多值索引

给数组类型的JSON创建索引

原JSON 数据

```
{ "title":"mybook", "author":"ROGER" }
```

创建一个函数索引

```
create index authindex1
  on mytable m
(m.jsontext.author.string())
```

如果需要创建索引的key的值是数组呢？

```
{
    "title":"mybook",
    "author": [ "ROGER", "JOHN" ]
}
```

利用多值索引…

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

多值索引 – 示例

多值索引的例子

创建表

```
create table mytable (jsoncol json);
insert into mytable values (
  '{ "title":"mybook", "author": [ "ROGER", "JOHN" ] }');
```

接下来创建 **multivalue index**

```
create multivalue index authindex2
  on mytable m(m.jsoncol.author.string());
```

查询

```
select * from mytable m
  where json_exists(jsoncol, '$?(@.author == "JOHN")');
```

- 查询条件的值类型必须与目标值类型一致
- 支支持 string(), number(), stringOnly() 或 numberOnly()
- Automatic conversion on first two, not on second two

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

多值索引 – 执行计划

多值索引的例子

```
---------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |              |     1 |  6104 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID BATCHED| MYTABLE      |     1 |  6104 |     2   (0)| 00:00:01 |
|   2 |   HASH UNIQUE                        |              |     1 |  6104 |            |          |
|*  3 |    INDEX RANGE SCAN (MULTI VALUE)    | AUTHINDEX2 |     1 |       |     1   (0)| 00:00:01 |
---------------------------------------------------------------------------------------------------
```

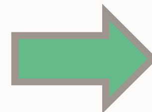

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

SQL/JSON 增强

JSON-like 语法增强

以非常直观的方式创建JSON 类型的结果

```
SELECT JSON {
  'empno' : empno,
  'name' : ename,
  'salary' : sal,
  'department' : (
    SELECT JSON {
      'name' : dname,
      'loc' : loc
    }
    FROM dept d
    WHERE d.deptno = e.deptno
  )
}
FROM emp e
WHERE e.empno = 7839;
```



```
{
  "empno" : 7839,
  "name" : "KING",
  "salary" : 5000,
  "department" : {
    "name" : "ACCOUNTING",
    "loc" : "NEW YORK"
  }
}
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

JSON_TRANSFORM

JSON_TRANSFORM 操作

JSON_Transform

- 遵循现有SQL / JSON语法和语义的新SQL / JSON运算符
- 将多个操作融合进一个操作中 (例如. 更新, 添加, 删除)
- 支持局部更新
- JSON_Transform 依托于原生 JSON 类型的优化，从而降低日志大小

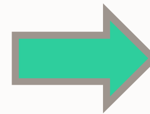
Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

JSON_TRANSFORM

JSON_TRANSFORM 示例 – 临时修改数据

```
select id,
json_serialize(json_data pretty) as data
from t1;
```

```
1, {
     "fruit" : "apple",
     "quantity" : 10
}
2, {
     "produce" : [ {
                    "fruit" : "apple",
                    "quantity" : 10
                   }, {
                     "fruit" : "orange",
                     "quantity" : 15
                  } ]
}
```



```
select json_transform(json_data,
                      set '$.quantity' = 20
                      returning clob pretty)
as data
From t1 where  id = 1;
```

```
1, {
     "fruit" : "apple",
     "quantity" : 20
}
2, {
     "produce" : [ {
                    "fruit" : "apple",
                    "quantity" : 10
                   }, {
                     "fruit" : "orange",
                     "quantity" : 15
                  } ]
}
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

JSON 数据更新

JSON_TRANSFORM 示例 – 临时增加数据

```
select
json_transform(json_data,
        insert '$.updated_date' = systimestamp
        returning clob pretty) as data
from t1
where id = 1;
```


```
{
 "fruit" : "apple",
 "quantity" : 10,
 "updated_date" : "2021-03-05T10:41:40.867246Z"
}
```

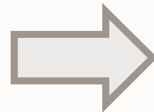

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

JSON 数据更新

JSON_TRANSFORM 示例 – 修改（复合操作）

```
UPDATE employees e
SET e.jcol =  JSON_TRANSFORM (e.jcol,
                   SET    '$.job' = 'Surfing Instructor',
                   SET    '$.salary' = (e.jcol.salary / 3),
                   REMOVE '$.phones[*]?(@.type == "work")',
                   APPEND '$.children' = 'Fiona’)
WHERE e.jcol.id = 123;
```

```
{
 "id” : 123,
 ”name” : "John Smith",
 "age” : 25,
 "job” : "Programmer",
 "salary” : 60000,
 "phones” :
   [ {"type” : "mobile",
      "number” : "555-123-4567"},
       {"type” : "work",
     "number” : "555-999-1111"} ],
 "children” :["Sydney","Sierra"]
}
```



```
{
 "id” :123,
 "name" : "John Smith",
 "age " : 25,
 "job " : "Surfing Instructor",
 "salary” : 20000,
 "phones” :
   [ {"type” : "mobile",
     "number” : "555-123-4567"} ],
 "children” : ["Sydney","Sierra","Fiona"]
}
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

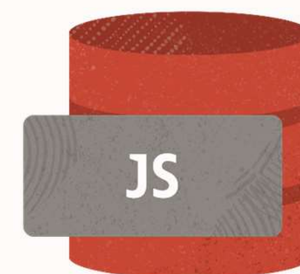
Database 中执行 Javascript

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

Oracle DB 中执行 Javascript

Multilingual Engine (MLE) runs data processing logic in JavaScript

- 在数据所在的 Oracle Database 内部执行Javascript
- JavaScript数据类型自动映射到Oracle Database数据类型，反之亦然
- 在PL / SQL和JavaScript之间无缝交换数据
- JavaScript代码本身可以通过内置JavaScript模块执行PL / SQL和SQL
- 使开发人员能够有效地使用现代编程语言工作



 Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

Oracle DB 中执行 Javascript

如何使用Javascript

如何在PL/SQL 中执行Javascript

```
DECLARE
  ctx DBMS_MLE.context_handle_t := DBMS_MLE.create_context();
BEGIN
  DBMS_MLE.eval(ctx, 'JAVASCRIPT', q'~console.log("Hello, World!");~');
  DBMS_MLE.drop_context(ctx);
END;
/
```

1. 创建JS执行上下文
2. 在上下文中执行JS 代码
3. 清理JS上下文

默认的，Javascript 的函数 console.log() 会将内容写入 PL/SQL package 的 DBMS_OUTPUT 缓冲区中

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

Oracle DB 中执行 Javascript

Javascript 执行上下文

- 利用 DBMS_MLE Package 的 create_context() 创建
- 对于创建多少个上下文没有限制
- 每个上下文都代表各自独立的运行时
- 上下文是有代价的（每个上下文都会消耗当前session中的内存）

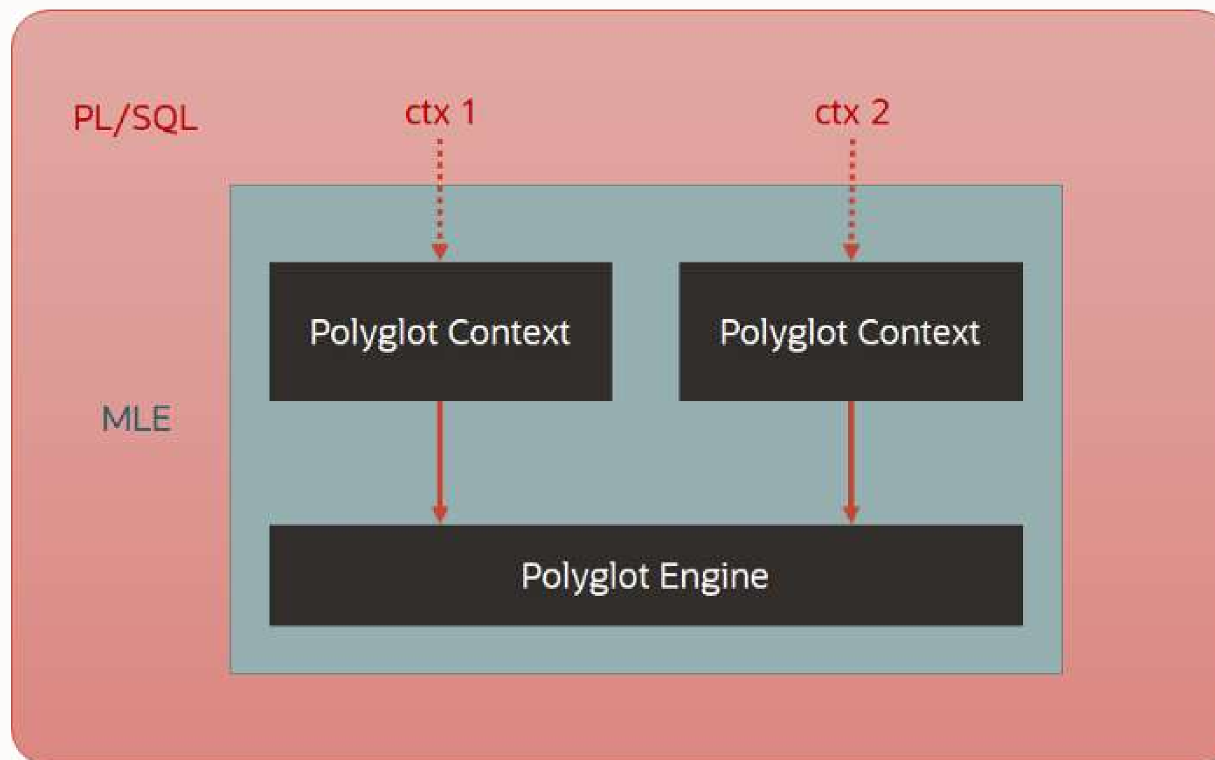
建议：

- 不要创建过多的context
- 在不用的时候，马上释放context

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

Oracle DB 中执行 Javascript

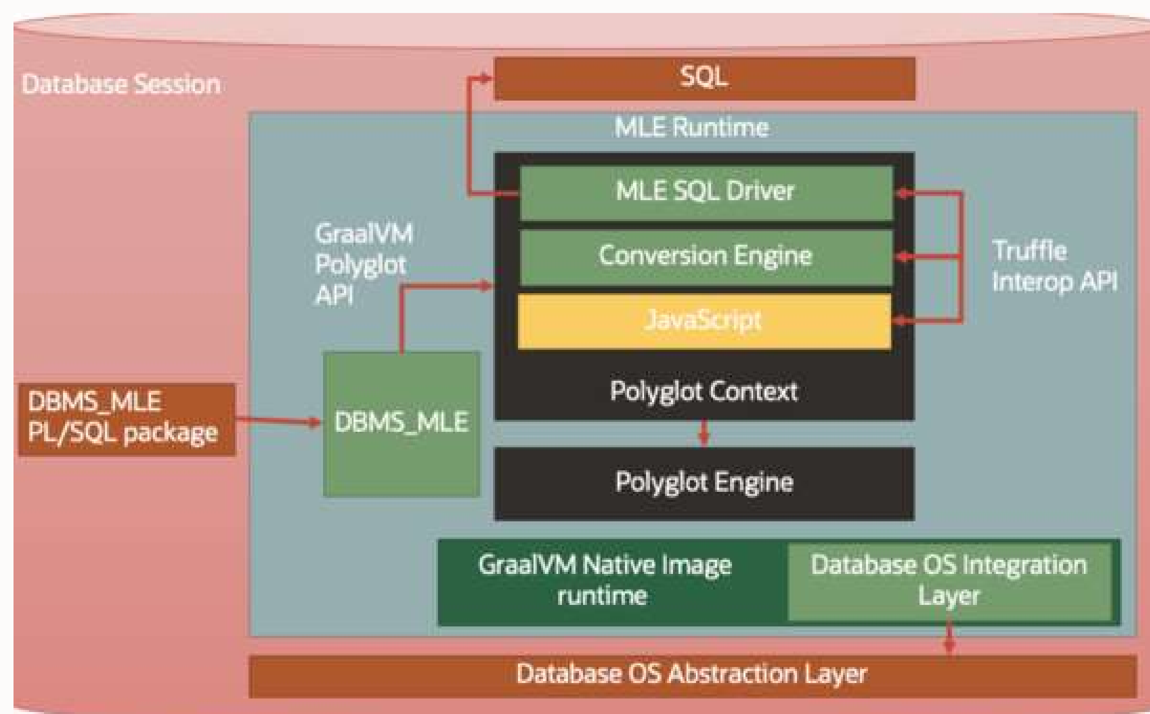
Javascript 执行引擎架构图



Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

Oracle DB 中执行 Javascript

Javascript 执行引擎架构图



Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

Oracle DB 中执行 Javascript

Javascript 与 PL/SQL 的值传递

```
DECLARE
  ctx dbms_mle.context_handle_t;
  source CLOB;
  greeting VARCHAR2(100);
BEGIN
  ctx := dbms_mle.create_context();
  dbms_mle.export_to_mle(ctx, 'person', 'World'); -- Export value from PL/SQL

  source := q'~
    var bindings = require("mle-js-bindings");
    var person = bindings.importValue("person"); // Import value previously exported from PL/SQL
    var greeting = "Hello, " + person + "!";
    bindings.exportValue("greeting", greeting); // E
  ~';

  dbms_mle.eval(ctx, 'JAVASCRIPT', source); -- Eval                    context
  dbms_mle.import_from_mle(ctx, 'greeting', greeting); -- Import value previously exported from MLE
  dbms_output.put_line('Greetings from MLE: ' || g
  dbms_mle.drop_context(ctx); -- D
END;
/
```

导出PL/SQL中的值

在Javascript中导入 PL/SQL中的值

将值导出到PL/SQL

执行Javascript

导入值

输出值

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

JDBC 对 JSON 数据类型的支持

支持原生JSON

- 从 Java 读取、写入和修改 JSON 类型值的工具
- JDBC 包 oracle.sql.json
- 特性
  - 可变的树/对象模型
  - 基于流的解析器和生成器
  - 访问扩展的 SQL/JSON 类型（TIMESTAMP、DATE 等）
  - 与 JSON-P 和 JSON-B 集成
  - 如何获取最新驱动?

    JAR:objdbc8.jar or ojdbc11.jar from database installation, OTN, or Maven Central Repository

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

JDBC 中 oracle.sql.json包

Description	Classes/interfaces
Tree model	OracleJsonObject OracleJsonArray OracleJsonString OracleJsonDecimal OracleJsonDouble OracleJsonTimestamp OracleJsonBinary OracleJsonIntervalDS OracleJsonIntervalYM ...
Event model	OracleJsonParser OracleJsonGenerator
Factory	OracleJsonFactory

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

OracleJsonFactory 工具类

Description	Methods

Read/write OSON	`createJsonBinaryGenerator(OutputStream)`
	`createJsonBinaryValue(ByteBuffer)`
	`createJsonBinaryValue(InputStream)`
	`createJsonBinaryParser(ByteBuffer)`
	`createJsonBinaryParser(InputStream)`

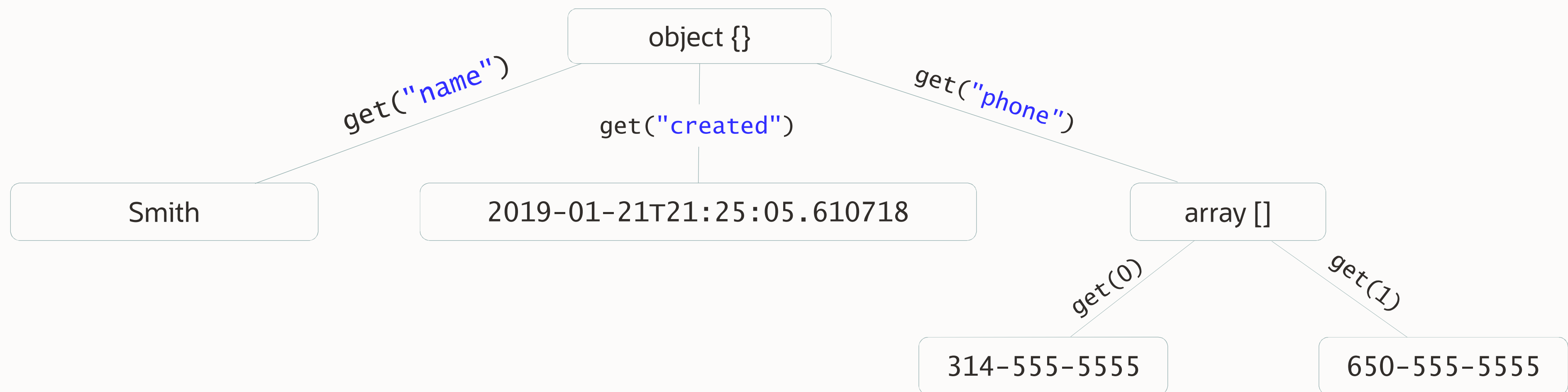
Read/write JSON text	`createJsonTextGenerator(OutputStream)`
	`createJsonTextGenerator(Writer)`
	`createJsonTextValue(InputStream)`
	`createJsonTextValue(Reader)`
	`createJsonTextParser(InputStream)`
	`createJsonTextParser(Reader)`

In-memory, mutable, tree model creation	`createObject(), createObject(OracleJsonObject)`
	`createArray(), createArray(OracleJsonArray)`
	`createString(String)`
	`createDecimal(BigDecimal), createDecimal(int), createDecimal(long)`
	`createDouble(double)`
	`createTimestamp(Instant)`
	`....`

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

树形结构

```
{
  "name" : "Smith",
  "created" : "2019-01-21T21:25:05.610718",
  "phone" : ["314-555-5555","650-555-5555"]
}
```



Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

例子：创建 JSON 对象

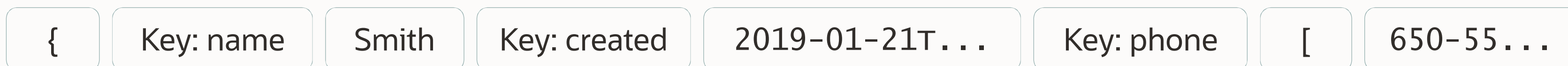
```
OracleJsonFactory factory = new OracleJsonFactory();

OracleJsonObject emp = factory.createObject();
emp.put("name", "Smith");
emp.put("salary", 40000);
emp.put("created", Instant.now());

System.out.println(emp.get("name"));
System.out.println(emp.toString());
```

```
Smith
{"name":"Smith","salary":40000,"created":"2019-01-21T21:25:05.610718"}
```

 Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

基于事件的模型

```
{
  "name" : "Smith",
  "created" : "2019-01-21T21:25:05.610718",
  "phone" : ["314-555-5555","650-555-5555"]
}
```



OracleJsonParser

- Produces a sequence of events
- `parser.next()`
- Events can come from text or OSON

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

生成 JSON 对象

```
OracleJsonFactory factory = new OracleJsonFactory();
FileOutputStream out = new FileOutputStream("emp.oson");
OracleJsonGenerator generator = factory.createJsonBinaryGenerator(out);
generator.writeStartObject();
generator.write("name", "smith");
generator.write("salary", 40000);
generator.write("created", Instant.now());
generator.writeEnd();
generator.close();
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

解析 JSON

```java
FileInputStream in = new FileInputStream("emp.oson");
OracleJsonParser parser = factory.createJsonBinaryParser(in);
while (parser.hasNext()) {
  switch (parser.next()) {
  case START_OBJECT:
     System.out.println("Start object.");
     break;
  case END_OBJECT:
     System.out.println("End object.");
     break;
  case KEY_NAME:
     System.out.println("Key: " + parser.getString());
     break;
  case VALUE_DECIMAL:
     System.out.println("Number: " + parser.getInt());
     break;
  case VALUE_STRING:
     System.out.println("String: " + parser.getString());
     break;
  case VALUE_TIMESTAMP:
     System.out.println("Timestamp: " + parser.getInstant());
     break;
  }
}
```

```
Start object.
Key: name
String: smith
Key: salary
Number: 40000
Key: created
Timestamp: 2019-01-21T21:53:39.833976Z
End object.
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

支持 JSON-P

- 应用程序已经实现了 javax.json 接口，并且您希望数据库使用它们
- 应用程序使用实现 javax.json 的第三方库，并且您需要数据库来使用它们您希望使
- 应用程序独立于 Oracle 数据库特定的 API 和扩展类型
- 需要使用 JSON-B

```
ResultSet rs = stmt.executeQuery("SELECT data FROM emp ");
rs.next();
JsonObject smith = rs.getObject(1, javax.json.JsonObject.class);
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

支持 JSON-B

```
public class Emp {
    String name;
    String job;
    BigDecimal salary;
}
...
YassonJsonb jsonb = (YassonJsonb) JsonbBuilder.create()
stmt = con.prepareStatement("SELECT e.data FROM emp");

ResultSet rs = stmt.executeQuery();
rs.next();
JsonParser parser = rs.getObject(1, JsonParser.class);
Emp e = jsonb.fromJson(parser, Emp.class);
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

SQL 宏

SQL 宏提供了一个易于封装复杂SQL 表达式的途径

- 类似于SQL 预处理.对于 SQL Engine 是可见的
- 轻松创建可重用和可移植的代码
- 简化常见SQL表达式的调用
- 无需昂贵的上下文切换

```
create function total_paid(
   quantity integer,unit_price number
)
  return varchar2
  sql_macro (scalar) as
begin
  return ‘ quantity * unit_price ’;
end total_paid;
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

SQL 宏

PL/SQL 函数 - 封装降低编写SQL语句的难易度

```
select *
  from order_items
 Where quantity * unit_price * 0.8 >
:total
```

原始SQL 逻辑一目了然,但不灵活

```
Select *
From order_titems
Where total_paid (
  quantity, unit_price
) > :total
```

利用PL/SQL函数编写 优雅且逻辑一目了然

但：太慢了

```
create or replace function total_paid(
               quantity number,
               unit_price number)
return number as pragma udf;
begin
       Return quantity * unit_price;
End total_paid;
```

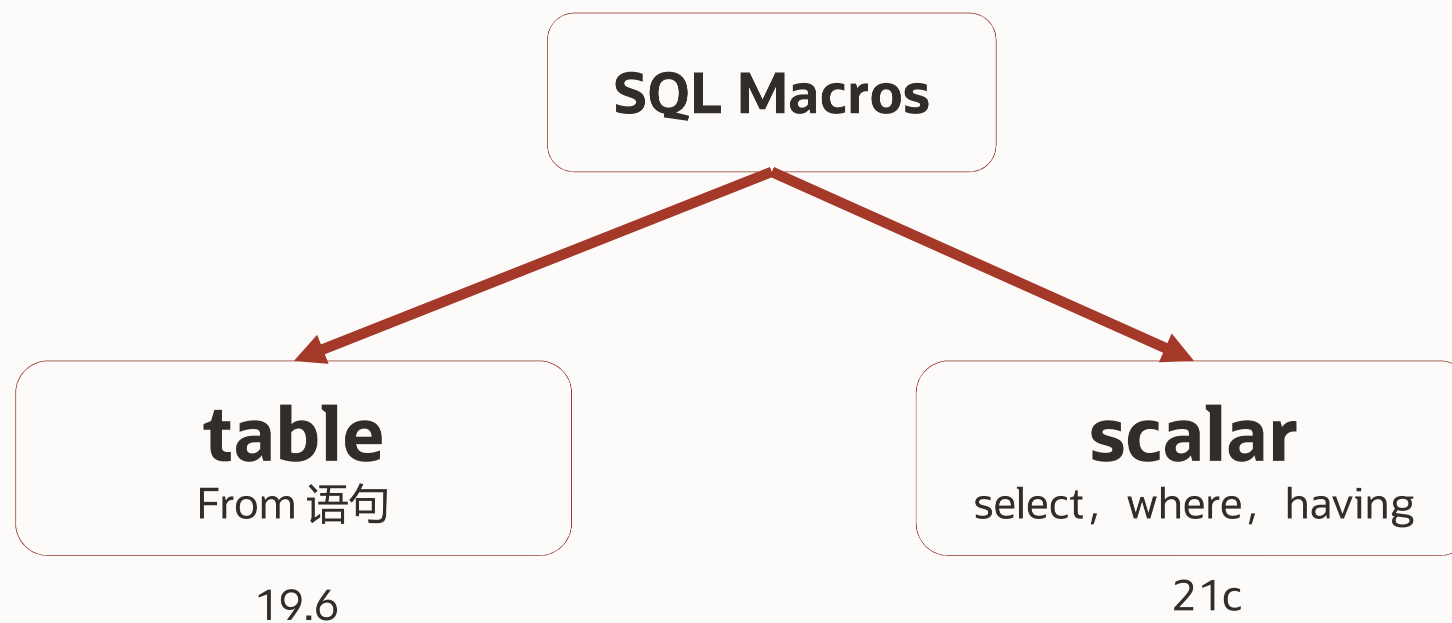
理由：

- SQL 与 PL/SQL 的执行引擎不同
- 每次条用函数，都需要切换上下文
- PL/SQL 函数对于 SQL 优化器是“黑洞”

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

SQL 宏

SQL Macros 的类型



FUNCTION RETURNS STRING

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

SQL 宏作为表的应用

SQL Table Macro – 封装FROM 语句中的表达式

- 使用在 from 语句中
- 类似于 inline view
- 可以接收 “表” 作为参数

```
create or replace function sal_by_dept
  return varchar2 sql_macro(table)
is
begin
  return q'{
    select deptno, sum(sal) as sal_tot
    from   emp
    group by deptno
  }';
end;
/
```

```
create or replace function row_count
(p_tab dbms_tf.table_t)
  return varchar2 sql_macro(table)
is
begin
  return q'{
    select count(*) as row_count from
p_tab
  }';
end;
/
```

 Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

SQL 宏作为表的应用

SQL Table Macro – 封装FROM 语句中的表达式

- 使用场景
  - 带参数的 VIEW
  - 动态表

```
CREATE OR REPLACE FUNCTION budget
return varchar2 SQL_MACRO
IS
BEGIN
  RETURN q'{ select department_id, sum(salary) budget
             from hr.employees
             group by department_id }';
END;
/
```

```
SQL> SELECT * FROM budget() WHERE department_id IN (10,50);
```

```
CREATE OR REPLACE FUNCTION budget_per_job(job_id varchar2)
return varchar2 SQL_MACRO
IS
BEGIN
  RETURN q'{ select department_id, sum(salary) budget
             from hr.employees
             where job_id = budget_per_job.job_id
             group by department_id }';
END;
/
```

```
SQL> SELECT * FROM budget_per_job('SH_CLERK') WHERE department_id = 50;
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

SQL Macros for Scalar

SQL 宏提供了一个易于封装复杂SQL 表达式的途径

如何使用

```
create or replace function
       sales_tax(unit_cost number,
                 unit_type varchar)
return varchar2 SQL_MACRO(SCALAR) is
begin
return q'{case when unit_type = 'FOOD'
          then unit_cost
          else unit_cost * 1.2 end}';
end;
```

- 必须返回字符串
- SQL scalar SQL macro can't have table arguments
- 若在PL/SQL中使用 SQL Macro 则直接返回字符串

```
SQL> select sales_tax(20,'WINE') from dual;

SALES_TAX(20,'WINE')
--------------------
                  24
```

```
SQL> select case
            when 'WINE' = 'FOOD' then 20
            else 20*1.2
        end
     from dual;
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

SQL Macros for Scalar

SQL Scalar Macro

- 使用在 select， where， Having 语句
- 不能接收表为参数

```
create or replace function
show_date(p_value  date)
  return varchar2 sql_macro(scalar)
is
begin
  return q'{ to_char(p_value, 'YYYY-MM-
DD') }';
end;
/
```

```
select show_date(sysdate) as my_date from
dual;




MY_DATE
----------
2020-12-26
```

 Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

SQL 宏

SQL Macro 包含在 Package 中

SQL Macro 可以像常规函数一样，包含在Package中

```
create or replace package date_macros as

  function show_date(p_value  date)
    return varchar2 sql_macro(scalar);

  function show_datetime(p_value  date)
    return varchar2 sql_macro(scalar);

  function show_timestamp(p_value  timestamp)
    return varchar2 sql_macro(scalar);

end;
/
```

```
select date_macros.show_date(sysdate) as my_date
from dual;

MY_DATE
----------
2020-12-26

select date_macros.show_datetime(sysdate) as
my_datetime from dual;

MY_DATETIME
-------------------
2020-12-26 21:06:41

select date_macros.show_timestamp(systimestamp) as
my_timestamp from dual;

MY_TIMESTAMP
-----------------------------
2020-12-26 21:07:19.438043
```


Copyright © 2021, Oracle and/or its affiliates. All rights reserved.

Thank you

Guang Xu
Oracle SE HUB Data Management

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.