



# Gentle introduction to SQL Macros in Autonomous Database

# About me....



## Keith Laker

Product Manager for Analytic SQL and Autonomous DW

**Oracle**

Blog: [oracle-big-data.blogspot.com](http://oracle-big-data.blogspot.com)

Twitter: [@ASQLBarista](https://twitter.com/ASQLBarista) [@AutonomousDW](https://twitter.com/AutonomousDW)

Email: [keith.laker@oracle.com](mailto:keith.laker@oracle.com)

# Agenda

- What is a SQL Macro?
- SQL Macros – simple examples of TABLE type SQL Macros
- Parameterized views using SQL Macros
  
- Wrap up

# What is a SQL Macro?

- SQL Macros are new in **Database 20c**
- Allows SQL developers to encapsulate complex processing within a macro which then be used anywhere inside SQL statement
- Two types of SQL Macros ->
  1. **TABLE** expressions used in a FROM-clause  
and coming soon...
  2. **SCALAR** expressions used in SELECT list, WHERE/HAVING, GROUP BY/ORDER BY clauses

# What is a **TABLE** type SQL Macro?

- Two distinct types:
  1. *Parameterized Views*
  2. *Polymorphic Views*

# What is a **Parameterized View**?

- Tables used in queries are fixed inside the definition of macro
- Arguments are passed in to select rows from those tables
- "shape" of queries returned is (typically) fixed.
- Common use of these parameterized views is when the scalar arguments are used to select a subset of the rows that are then aggregated

# Parameterized View – Simply pass in arguments

```
CREATE FUNCTION budget (dept_no number
DEFAULT 10)
RETURN varchar2 SQL_MACRO(TABLE)
IS BEGIN
  RETURN q'[
    SELECT
      d.deptno,
      SUM(e.sal) budget,
      ANY_VALUE(d.dname) department,
      count(e.empno) headcount,
      count(e.mgr) mgr_headcount
    FROM emp e, dept d
    WHERE d.deptno = :dept_no
    AND e.deptno = d.deptno
  GROUP BY d.deptno]';
end BUDGET;
/
```

```
WITH east_coast as
(SELECT deptno
 FROM dept
 WHERE loc = 'Boston')

SELECT *
FROM budget(east_coast);
```



# What is a Polymorphic View?

- Table valued macros that have one or more table arguments
  - can additionally have scalar valued arguments as well!
- Input tables are used inside the query returned by macro.
- Example: anti-select where...
  - for a given table return a query that skips columns of a given name or data-type
  - Pass in generic predicates (e.g. rownum < n),
  - Provide a functional syntax for existing syntax

```
select *  
from NJ(sales, products, customers)
```



```
select *  
from sales  
natural join products  
natural join customers
```



# Dynamic SQL Macros – Polymorphic Views

```
CREATE OR REPLACE FUNCTION row_sampler(t
DBMS_TF.Table_t, pct number DEFAULT 5)
    RETURN VARCHAR2 SQL_MACRO(TABLE)
AS
BEGIN
    RETURN q' {SELECT *
                FROM t
                order by dbms_random.value
                fetch first row_sampler.pct percent rows
only} ' ;
END;
/
```

# Dynamic SQL Macros – Polymorphic Views

```
SELECT *  
FROM row_sampler(t=>sh.sales, pct=>15);
```

# Using SQL Macros (TABLE) to Return a Range

- Generates an arithmetic progression of rows in the range [first, stop).
  - First row will have the value first
  - Each subsequent row's value will be step more than previous row's value
- Semantics of these macros are modeled after the Python's built-in range() function
  - PL/SQL Package
  - PL/SQL Package Body

# Using SQL Macros to Return a Range

```
create or replace package GEN is
```

```
    function gRANGE(gstop number)
        return varchar2 SQL_MACRO(TABLE);
```

```
    function gRANGE(gfirst number default 0, gstop number, gstep number default 1)
        return varchar2 SQL_MACRO(Table);
```

```
end GEN;
```

```
/
```

# Using SQL Macros to Return a Range – *Function Overloading*

```
create or replace package body GEN is
  function gRANGE(gstop number)
    return varchar2 SQL_MACRO(TABLE) is
begin
  return q'{
    select * from (select level-1 n from dual connect by level<=gstop) where gstop>0
  }';
end;

function gRANGE(gfirst number default 0, gstop number, gstep number default 1)
  return varchar2 SQL_MACRO(TABLE) is
begin
  return q'{
    select gfirst+n*gstep n from grange(round((gstop-gfirst)/nullif(gstep,0)))
  }';
end;

end GEN;
/
```

**Note: We have SQL Macro calling SQL Macros – NESTED Macros**

# Examples - Using SQL Macros to Return a Range

```
SQL> select * from gen.range(5);
```

```
-----  
N  
-----  
0  
1  
2  
3  
4
```

```
SQL> select * from gen.range(0, 1, step=>0.1);
```

```
-----  
N  
-----  
0  
.1  
.2  
.3  
.4  
.5  
.6  
.7  
.8  
.9
```

```
SQL> select * from gen.range(5, 10);
```

```
-----  
N  
-----  
5  
6  
7  
8  
9
```

```
SQL> select * from gen.range(+5,-6,-2);
```

```
-----  
N  
-----  
5  
3  
1  
-1  
-3  
-5
```

# Updates to Dictionary Views

- {USER, ALL, DBA}\_PROCEDURES views will have new column called SQL\_MACRO.
  - Column can have the values NULL, SCALAR, or TABLE.
- Following SQM functions were created with the annotation SQL\_MACRO, SQL\_MACRO(SCALAR), and SQL\_MACRO(TABLE) respectively:

OBJECT_NAME	SQL_MACRO	OBJECT_TYPE
SQL_MACRO_DDL	TABLE	FUNCTION
SQL_MACRO_DDL_S	SCALAR	FUNCTION
SQL_MACRO_DDL_T	TABLE	FUNCTION



# Summary

- SQL Macros offers a simple SQL-based framework for encapsulating business/technical logic
  - *Avoids the need to call custom PL/SQL procedures, functions within queries*
- Automatically inherits all the usual in-database query optimizations
- Makes it possible to build parameterized views!

ORACLE®