

Polymorphic Table Functions and Qualified Expressions

Robert Marz



SEATTLE

ODTUG
Kscope19 
SEATTLE, WASHINGTON • JUNE 23-27

**PLEASE FILL OUT
YOUR EVALUATIONS**

 Washington State
Convention Center

Robert Marz – Independent Consultant



Client

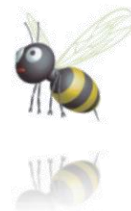
Senior Technical Architect
with database centric view of the world



ORACLE
ACE

its-people

Portfolio Manager Database Technologies
Blog Editor



DOAG (German Oracle User Group)

Active Member of Database Community
Responsible for Cloud Topics



@RobbieDatabee



blog.its-people.de



Robert.Marz
@its-people.de

450+ Technical Experts Helping Peers Globally

ORACLE[®]
ACE PROGRAM



ORACLE[®]
ACE Director



ORACLE[®]
ACE



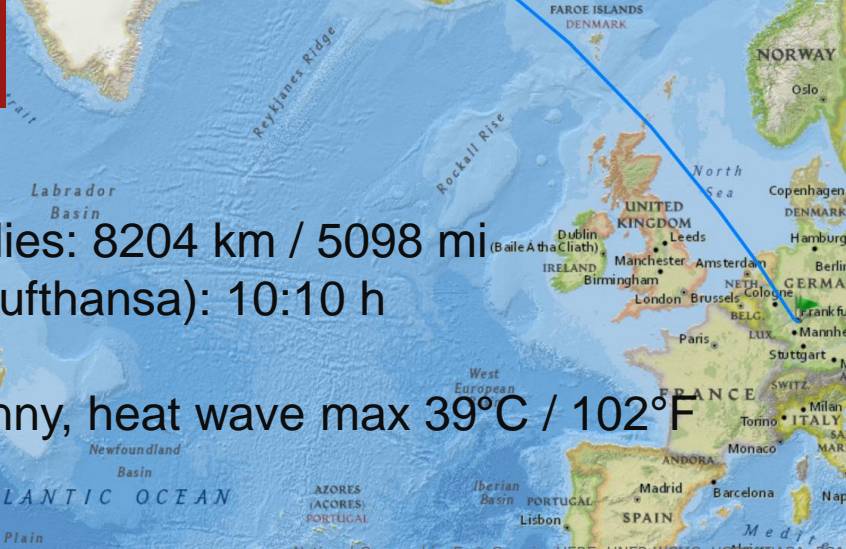
ORACLE[®]
ACE Associate

bit.ly/OracleACEProgram

Nominate yourself or someone you know: acenomination.oracle.com



Frankfurt, DE – Seattle, WA



Distance as the Crow flies: 8204 km / 5098 mi

Flight Time of Crane (Lufthansa): 10:10 h

Time Shift: -9 Hours

Weather: Frankfurt: sunny, heat wave max 39°C / 102°F

Motivation

A low-angle shot of a red-painted wooden ladder extending towards the top of the frame. The ladder is set against a bright blue sky filled with wispy white clouds. A person's hand is visible on the right side, firmly grasping one of the rungs. The ladder's rungs are horizontal, and the side rails are diagonal, creating a strong sense of upward movement and aspiration.

PL/SQL: Requirements for Web Development and Microservices

Microservices

PL/SQL

Data Formats

- JSON
- YAML

Business Logic

- Encapsulate
- APIs
- Re-Usability

constant improvements

Release 12:

- JSON Support

Release 18:

- Polymorphic Table Functions
- Qualified Expressions

Qualified Expressions



Qualified Expressions

Constructors
for PL/SQL
Types

Associative Arrays

Records

Initialize
Types

Function Call

Same Name as Type

Provide Values

By Name

By Position

Syntactical
Sugar

Shorter Code

Better readability


```
type numbers_ty is table of number  
    index by pls_integer;
```

```
type user_properties_ty is record(  
    is_ops boolean  
    , is_dev boolean  
    , email varchar2(255)  
);
```

```
type users_ty is table of user_properties_ty  
    index by varchar2(32);
```

```
numbers    numbers_ty;
appUsers   users_ty;
begin
numbers(2) := 10.5;
numbers(3) := 65;
numbers(1) := 3.14;

appUsers('alice').is_ops := false;
appUsers('alice').is_dev := true;
appUsers('alice').email  := 'alice@ioug.org';

appUsers('bob').is_ops := true;
```

-- Qualified Expressions are Constructors

-- Initialization by Name

```
numbers    numbers_ty := numbers_ty(2 => 10.5, 3 => 65, 1 => 3.14);
```

```
appUsers  users_ty := users_ty(
```

```
  'alice'   => user_properties_ty(is_ops => false,
                                is_dev => true,
                                email  => 'alice@ioug.org'),
```

```
  'robbie'  => user_properties_ty(email  => 'robbie@doag.org',
                                is_dev => true,
                                is_ops => true ),
```

-- Initialization by Position

```
  'bob'     => user_properties_ty(true, false, 'bob@ioug.org'));
```

Index by pls_integer

Index by varchar2

Qualified Expressions: Demo



```
-- Qualified Expressions in 18c
declare
  type numbers_ty is table of number
    index by pls_integer;

  type user_properties_ty is record(
    is_ops boolean
  , is_dev boolean
  , email varchar2(255)
  );

  type users_ty is table of user_properties_ty
    index by varchar2(32);
  -- Qualified Expressions are Constructors for Types
  numbers numbers_ty := numbers_ty(2=>10.5, 3=>65, 1=> 3.14); --
Initialization by Name
  appUsers users_ty := users_ty(
    'alice' => user_properties_ty(is_ops => false, is_dev => true,
email=>'alice@rmoug.org')
    , 'bob'   => user_properties_ty(true, false, 'bob@rmoug.org') --
Initialization by Position
    , 'robbie' => user_properties_ty(email=>'robbie@doag.org', is_dev =>
true, is_ops => true )
    );
  usr varchar2(32);
begin
  for idx in 1..numbers.count loop
```

```
    dbms_output.put_line('Index: '||idx||' value: '||numbers(idx));
  end loop;

  usr := appUsers.first;
  while usr is not null loop
    dbms_output.put_line(case appUsers(usr).is_dev when true then 'Dev' else
' end
                        ||case appUsers(usr).is_ops when true then 'Ops '
                        ||' User: '||usr|| ' email: '||appUsers(usr).email
                        );
    usr := appUsers.next(usr);
  end loop;
end;
/
```

Demo

Not on 18c, yet? Try LiveSQL



Oracle Live SQL - Code Library

https://livesql.oracle.com/apex/?p=590.49.6516689200719

ORACLE Live SQL

Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
Code Library

Code Library

qualified

Area: All

Types: All, Tutorials, Scripts

Sort By: Date Added, Executions, Name, Likes

Results Per Page: 60

Reset Search

Qualified Expressions for Associative Arrays (aka, collection constructors)

Aggregates and their necessary adjunct, qualified expressions, improve program clarity and programme...

3 ❤️ 12 months ago
6 ▶️ Steven Feuerstein (Oracle)

SCRIPT 18c,collection,array,initialize,constructor

18c Assigning Values to RECORD Type Variables Using Qualified Expressions

This example shows the declaration, initialization, and definition of RECORD type variables.

0 ❤️ 11 months ago
4 ▶️ Oracle

SCRIPT 18c

Qualified Expressions for Records (aka, record constructors)

Aggregates and their necessary adjunct, qualified expressions, improve program clarity and programme...

1 ❤️ 12 months ago
3 ▶️ Steven Feuerstein (Oracle)

SCRIPT 18c,record,initialize

18c Assigning Values to Associative Array Type Variables Using Qualified Expressions

This example uses a function to display the values of a table of BOOLEAN.

0 ❤️ 11 months ago
1 ▶️ Oracle

SCRIPT 18c

Table Functions (Classic)



Classic Table Functions

Generate Data

arbitrary
number of Rows

Row Structure

defined at compile time
PL/SQL Types

Nesting Possible

Pipelining

PL/SQL Function

Package OK
arbitrary Parameters

-- Classic Table Functions (9i and up)

```
create or replace package tf
```

```
as
```

```
    type fibo_rec is record (fibo number, ind number, tmp number);
```

```
    type fibo_tab is table of fibo_rec;
```

```
    function fibonacci(fibolimes in number)
```

```
        return tf.fibo_tab pipelined;
```

```
end tf;
```

```
/
```



```
create or replace package body tf
as
    function fibonacci(fibolimes in number)
        return tf.fibo_tab pipelined
    is
        fibo fibo_rec; -- Type with column definition
    begin
        fibo.ind := 1; fibo.fibo := 1; -- Pre 18c Init
        while fibo.fibo <= fibolimes
        loop
            pipe row (fibo);
            fibo.tmp := fibo.ind + fibo.fibo;
            fibo.fibo := fibo.ind; fibo.ind := fibo.tmp;
        end loop;
    end fibonacci;
end tf;
```

```
create or replace package body tf
as
  function fibonacci(fibolimes in number)
    return tf.fibo_tab pipelined
  is
    fibo fibo_rec := fibo_rec(1,1,null); -- Qualified Expression
  begin
    -- fibo.ind := 1; fibo.fibo := 1; -- Pre 18c Init
    while fibo.fibo <= fibolimes
    loop
      pipe row (fibo);
      fibo.tmp := fibo.ind + fibo.fibo;
      fibo.fibo := fibo.ind; fibo.ind := fibo.tmp;
    end loop;
  end fibonacci;
end tf;
```

```
select fibo
  from table(tf.fibonacci(15));
```

```
-- Since Oracle 12.2
-- the table()-Operator
-- is obsolete
```

```
select fibo
  from tf.fibonacci(15);
```

FIBO

1

1

2

3

5

8

13

7 rows selected.

Polymorphic Table Functions



Polymorphic Table Functions (PTF)

Modify
Source Table

Add / Remove / Modify

Rows & Columns

Generic
Extension

Like a View
but more procedural

Works for arbitrary input
tables

Business
Logic

API for Analysts

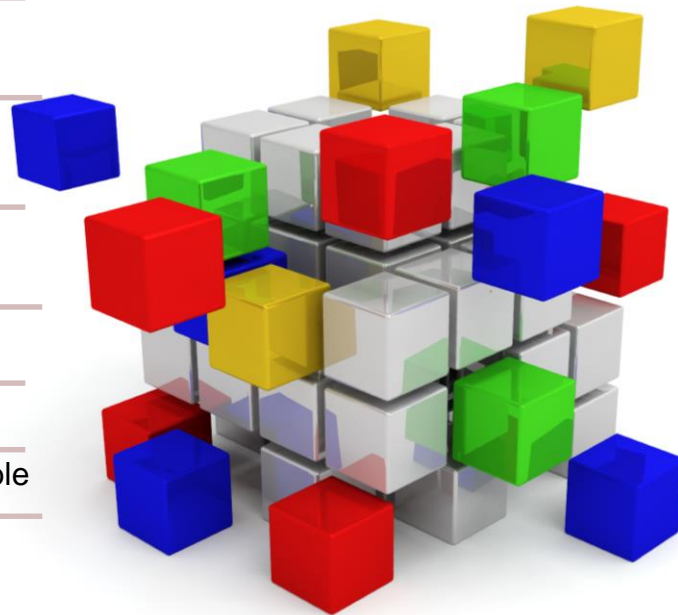
Hide complexity

make dynamic SQL available

SQL 2016
Standard

Not (yet) completely
implemented

Some (minor) discrepancies



PTF Benefits

Minimal data-
movement

Only columns of interest are
passed to PTF

Predicates, Projections,
Partitioning

pushed into underlying table/query
(where semantically possible)

Bulk data transfer

Into and out of PTF

Parallelism based on

type of PTF

query specified partitioning (if any)



D

E

M

O

Taking an existing rowset and...

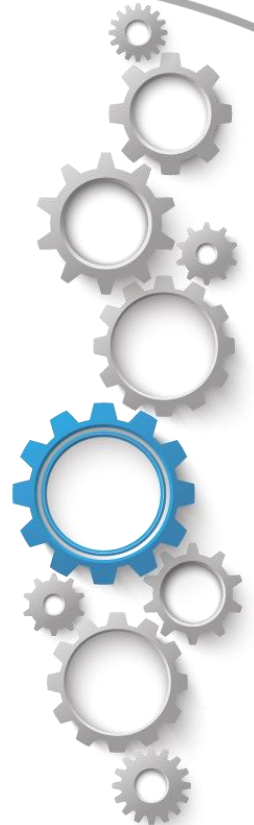
- Column-based **EXPANSION**
 - Calculating/deriving a new column value
- Row-based **EXPANSION**
 - Data pivot operation
- Column-based **REDUCTION**
 - Data unpivot operation
- Row-based **REDUCTION**
 - Data aggregation/reduction operation

No existing rowset to process...

- Rowset **GENERATOR**
 - Creates new rows and columns
 - Importing a CSV file

PTF Implementation

Input	Oracle: exactly one Table Column Lists arbitrary additional Parameters
Row Structure “Describe”	Fixed at SQL-Execution Time PL/SQL Function
Implementation	PL/SQL Package per PTF Heavy use of PL/SQL Tables DBMS_TF – Helper & Utilites
Performance	New Execution Plan operation Not always needed



What makes a PTF

Function	without	body	
	references	Package	<pre> create or replace function add_labels (tabname table, colnames columns) return table pipelined row polymorphic using labels; -- Package Name </pre>
Package	required	describe	
	function		
	optional	open	<pre> select * from add_labels(emp, columns(empno, mgr)); </pre>
	procedures	fetch_rows	
		close	

```
select *  
  from add_labels(emp, columns(empno, mgr) );
```

Variadic Pseudo Operators **columns()**

operates with a variable number(≥ 1) of operands

introduced in 18c to support PTF

can only appear in argument lists of PTF

parsed by SQL Engine

converted to corresponding DBMS_TF-types

passed as Input parameter to the DESCRIBE-Function

Describe Function

invoked by SQL-Engine at parse-time

determines the row_type

new & removed columns
returns dbms_tf.describe_t Table

marks columns for processing

“pass-through” – unchanged, not moved
“for read” - passed to fetch_rows procedure
via dbms_tf.table_t (IN OUT Parameter)



©3dexc - stock.adobe.com

Open & Close Procedures

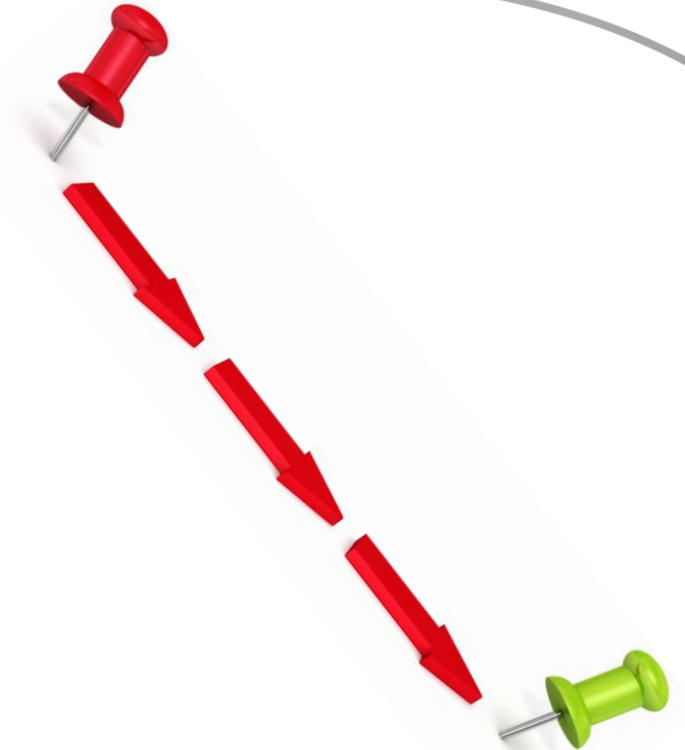
- Setup and Teardown of Environment
- Your instrumentation Code goes here
- Both are optional

Open

- called before first `fetch_rows` execution
- Initialize Variables

Close

- called at the end, after all `fetch_rows`
- do cleanup stuff



Fetch_rows is the worker

- processes rowsets (chunks of Table Data)
- only columns marked for read
- Needs same scalar parameters as PTF

Database can call multiple times

- for each rowset
- in parallel

Produce & Reduce Data

- fill new columns
- generate new rows
- suppress rows

©3dasec - stock.adobe.com



Polymorphic Table Functions: Flavors



PTF Semantic Types

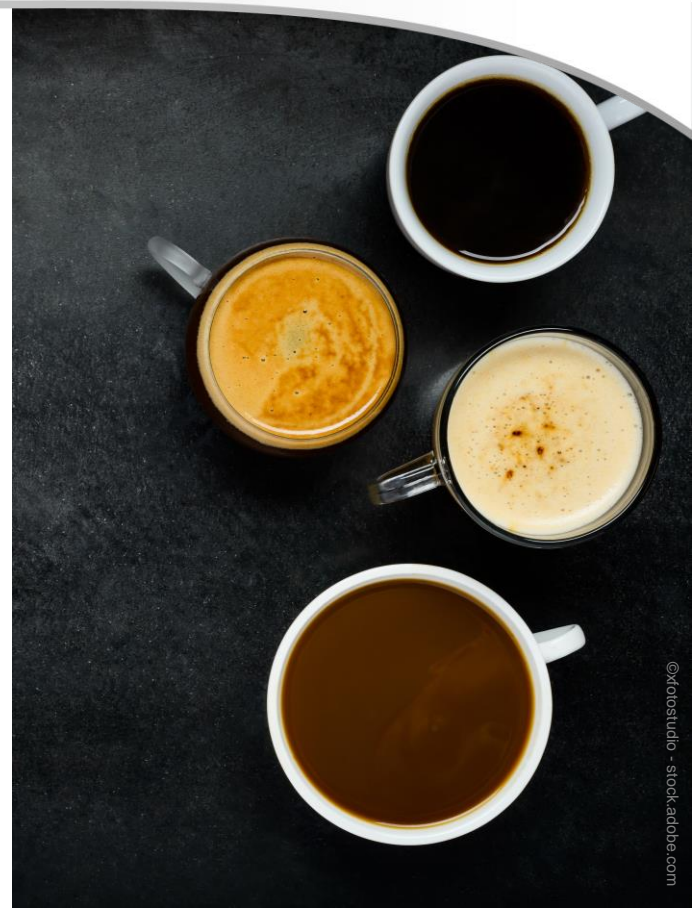
Determine execution Plan
Impact Performance

Row Semantic PTF

new columns can be derived from current row
return **table pipelined row** polymorphic

Table Semantic PTF

works on whole table or partition
return **table pipelined table** polymorphic



Datatype Restrictions

passthrough is possible with any type
“for read” and new columns must be scalar datatypes
`dbms_tf.supported_type` function

Invocation and Execution Restrictions

PTF cannot be nested in from clause
workaround: Use with-clause
PTF cannot be an argument to a (classic) table function
PTF yields no rowids
PARTITION BY and ORDER BY only work with Table Semantics PTF
DESCRIBE function cannot be called directly



**Want to know
more?**

[Database PL/SQL Language Reference](#)

[12.6 Overview of Polymorphic Table Functions](#)

[PL/SQL Packages and Types Reference](#)

[173 DBMS TF](#)

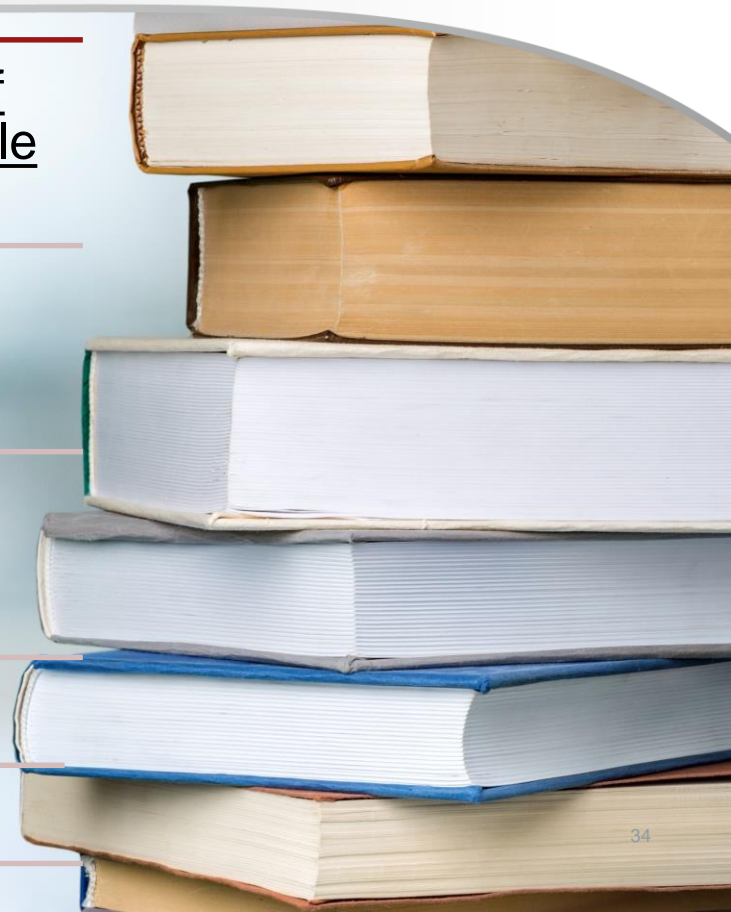
[LiveSQL](#)

[Code Library](#)

[Google](#)

[Blog Posts](#)

[Presentations](#)



Polymorphic Table Functions: LiveSQL



Oracle Live SQL - Code Library

https://livesql.oracle.com/apex/?p=590.49.6516689200719:

ORACLE Live SQL

- Home
- SQL Worksheet
- My Session
- Schema
- Quick SQL
- My Scripts
- Code Library**

Code Library

polymorphic

Area: All

Types: All, Tutorials, Scripts

Sort By: Date Added, Executions, Name, Likes

Results Per Page: 60

Reset Search

18c Echo Polymorphic Table Function

This PTF returns all the columns in the input table tab, and adds to it the columns listed in cols b...

SCRIPT 18c ARPLS

1 ❤️ 11 mo
22 ▶️ 👤 Oracle

Dynamic CSV to Columns Converter: Polymorphic Table Function Example

An example of how to use polymorphic table functions in 18c to dynamically convert CSV data to colum...

SCRIPT 18c, polymorphic table functions

5 ❤️ 12 mo
22 ▶️ 👤 Chris S

18c polymorphic table function TAB2KEYVAL

An example of using a polymorphic table function (PTF) to transpose columns to rows

SCRIPT 18c, PTF, UNPIVOT

2 ❤️ 12 mo
12 ▶️ 👤 Andrej

Polymorphic Table Function Split Column

A Polymorphic Table Function to split the first column of a table using ; as a separator

SCRIPT Polymorphic Table Function Split Column

1 ❤️ 6 week
12 ▶️ 👤 Patch7

Polymorphic Table Functions (PTFs) with variables

PTF's accept variables for use during parse or execution. This included scalar datatypes and PTF spe...

SCRIPT Polymorphic Table Function PTF Pseudo-Operator

0 ❤️ 35 6 week
12 ▶️ 👤 Darryl



D

E

M

O

```
create or replace package ptf_tags
as
    function describe(tabname in out dbms_tf.table_t,
                      colnames in dbms_tf.columns_t,
                      tag_string in varchar2)
        return dbms_tf.describe_t;

    procedure fetch_rows(tag_string in varchar2);
end ptf_tags;
/
```

Demo

```
create or replace
function add_tags(tabname table,
                  colnames columns,
                  tag_string varchar2)

return
  table pipelined
  row polymorphic
  using ptf_tags; -- Package Name
/
```

Demo

Demo: PTF addTags – Package Body – Function define



```
function describe(tabname in out dbms_tf.table_t, colnames in dbms_tf.columns_t, tag_string in varchar2)
    return dbms_tf.describe_t
as
new_cols dbms_tf.columns_new_t;
begin
    for i in 1 .. tabname.column.count -- loop over all table columns
    loop -- skip columns with unsupported data types
        continue when not dbms_tf.supported_type(tabname.column(i).description.type);
        for j in 1 .. colnames.count
        loop
            if (tabname.column(i).description.name = colnames(j)) -- is the column in the colnames table?
            then
                tabname.column(i).for_read := true;
                tabname.column(i).pass_through := true;
                new_cols(i) := tabname.column(i).description; -- copy column in new_cols()
                -- set datatype to varchar2
                new_cols(i).type := dbms_tf.type_varchar2;
                new_cols(i).max_len := 4000;
                new_cols(i).name := 'TAGGED_' -- set new column name
                    || regexp_replace(new_cols(i).name, '^"|"$') -- remove trailing or leading ',"'
                    || '„';
            end if;
        end loop;
    end loop;
    return dbms_tf.describe_t(new_columns => new_cols);
end describe;
```

Demo

Demo: PTF addTags – Package Body – Procedure fetch_rows



```
procedure fetch_rows(tag_string in varchar2)
as
  rowset dbms_tf.row_set_t;
  rowcount pls_integer;
  colcount pls_integer;
  tag_value varchar2(4000);
begin
  dbms_tf.get_row_set(rowset, rowcount, colcount);
  dbms_output.put_line(rowcount||' - '||colcount);
  for i in 1..rowset.count loop
    dbms_output.put_line(rowset(i).description.name||' - '||rowset(i).tab_varchar2.count||' -
' ||rowset(i).tab_number.count);
  end loop;
  for i in 1..rowcount loop
    for j in 1..colcount loop
      -- The new columns are varchar2(4000)
      tag_value:= case rowset(j).description.type
        when dbms_tf.type_varchar2 then rowset(j).tab_varchar2(i)
        when dbms_tf.type_number then to_char(rowset(j).tab_number(i))
        else 'DATATYPE NOT IMPLEMENTED'
      end;
      rowset(j).tab_varchar2(i) := replace(tag_string, '%s',tag_value);
    end loop;
  end loop;
  dbms_tf.put_row_set(rowset);
end fetch_rows;
```

Demo

Conclusion



Qualified Expressions

- more than syntactical sugar
- make code shorter
- improve readability

Polymorphic Table Functions

- Simplifies SQL for non-technical Users
- Great potential
- Powerful & flexible Enhancement to SQL

PLEASE

**DO
TRY THIS
AT HOME**

* or at [LiveSQL.oracle.com](https://livesql.oracle.com)



Vielen Dank für Ihre
Aufmerksamkeit.