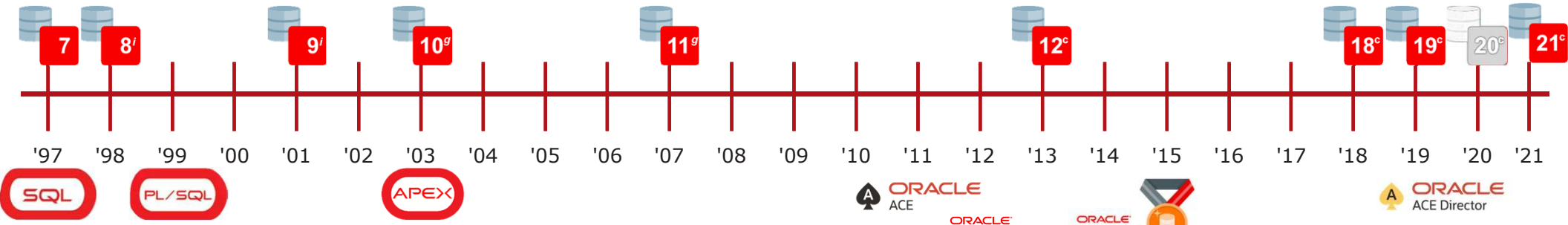
QUALOGY

PBarel@Qualogy.com
http://blog.bar-solutions.com

QUALOGY

About me…



ORACLE ACE

ORACLE ACE Director

ORACLE Certified Associate PL/SQL Developer

ORACLE Certified Professional Advanced PL/SQL Developer

bar-solutions.com blog http://blog.bar-solutions.com

All things ORACLE http://allthingsoracle.com

OTECH MAGAZINE http://www.otechmag.com

www.red-gate.com/simple-talk/author/patrick-barel/

bar-solutions.com/otechmagazine.php

Plugins for PL/SQL Developer
http://plugins.bar-solutions.com

Contact me…

@patch72

PBarel@Qualogy.com

Patrick.Barel@GMail.com

patrick@bar-solutions.com

Patrick.Barel@GMail.com

3029156

40338721

Patrick Barel

ORACLE
ACE Program

500+ technical experts helping peers globally

The Oracle ACE Program recognizes and rewards community members for their technical contributions in the Oracle community

3 membership tiers

ORACLE ACE Director | ORACLE ACE | ORACLE ACE Associate

For more details on Oracle ACE Program:
bit.ly/OracleACEProgram

Oracle Groundbreakers

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

Connect: oracle-ace_ww@oracle.com Facebook.com/oracleaces @oracleace

Oracle Cloud Infrastructure

New Free Tier

oracle.com/cloud/free

Always Free

Services you can use for unlimited time

+

30-Day Free Trial

Free credits you can use for more services

A Gentle Introduction to Polymorphic Table Functions

Patrick Barel, Qualogy

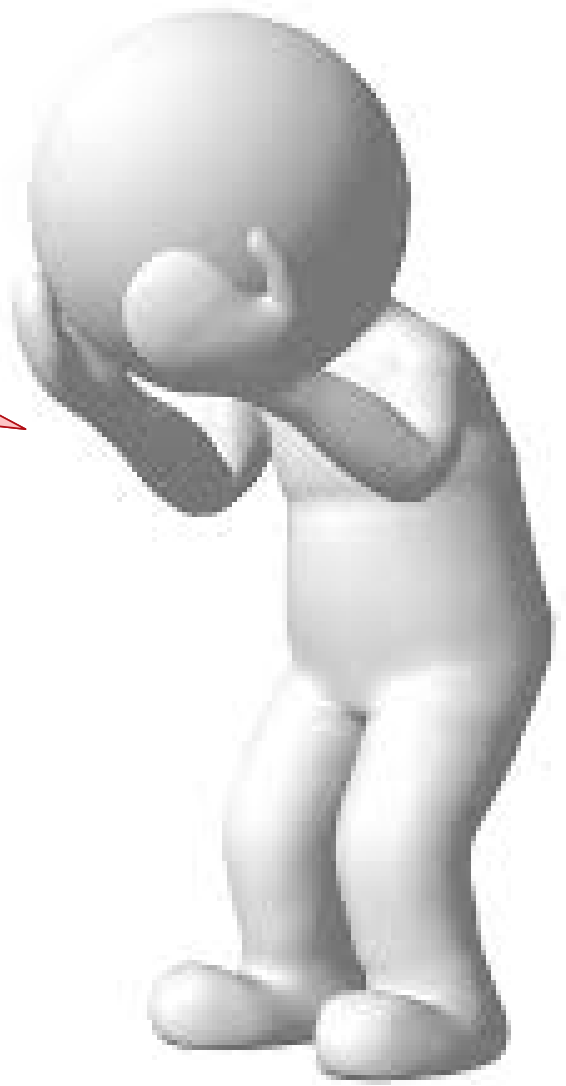
October 13, 2021

QUALOGY

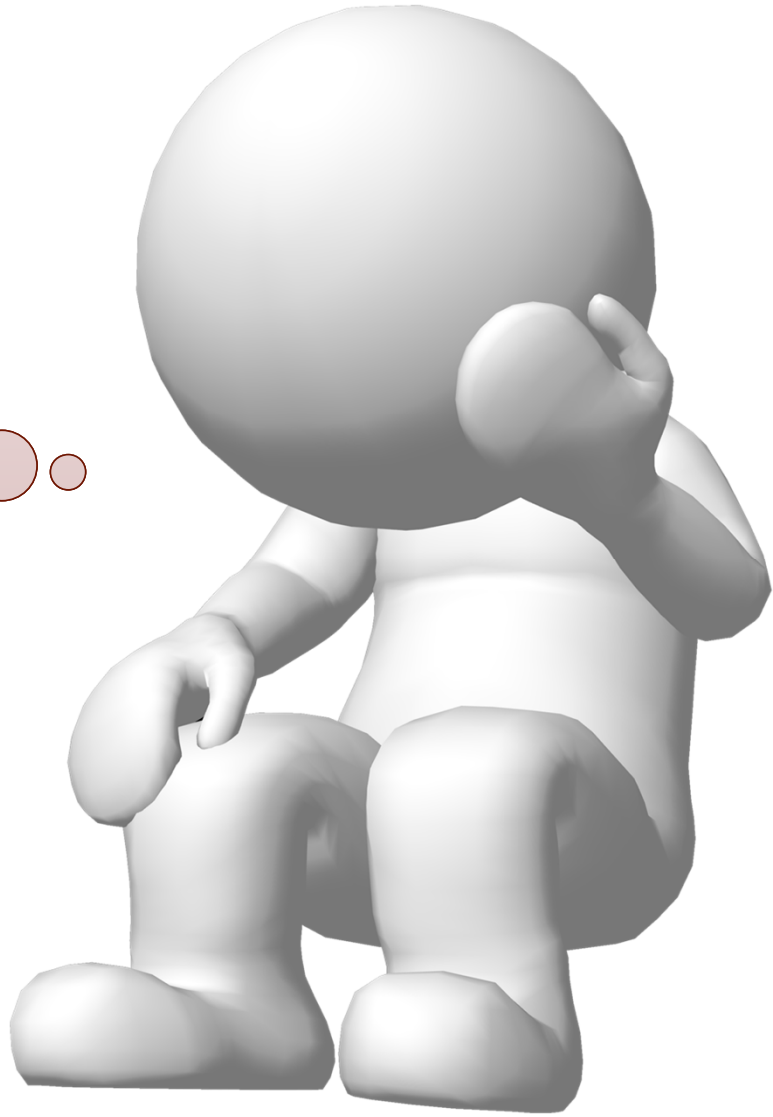

ORACLE
OPEN
WORLD
Oracle OpenWorld 2017
October 1-5, 2017
San Francisco, CA
www.oracle.com/openworld
ORACLE

Can you tell me in Tom and Jerry terms what POLYMORPHIC TABLE FUNCTIONS are?

Are you out of your mind?
The ANSI documentation alone is a 200 page document



What ARE Polymorphic Table Functions

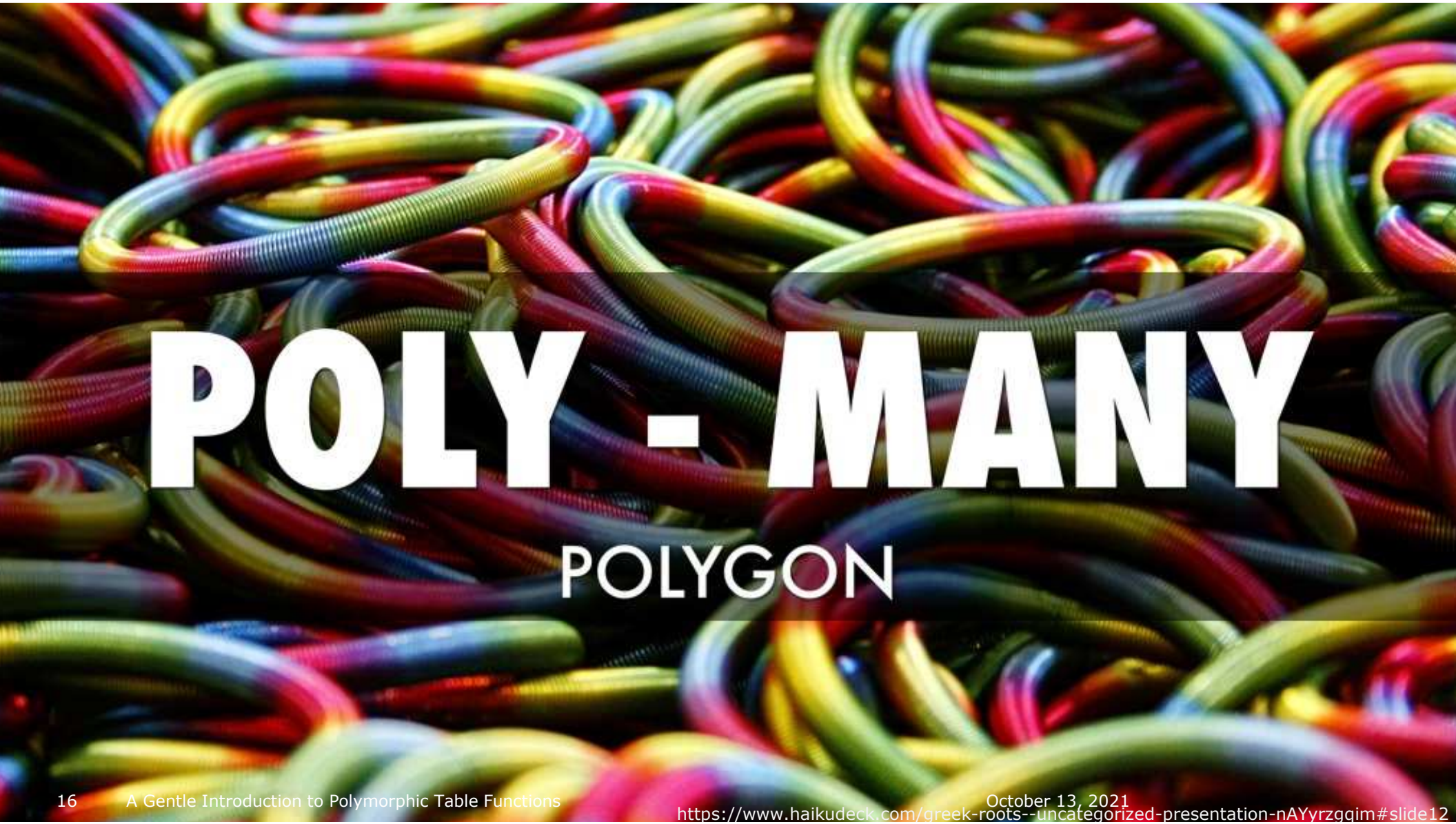



Google


Google

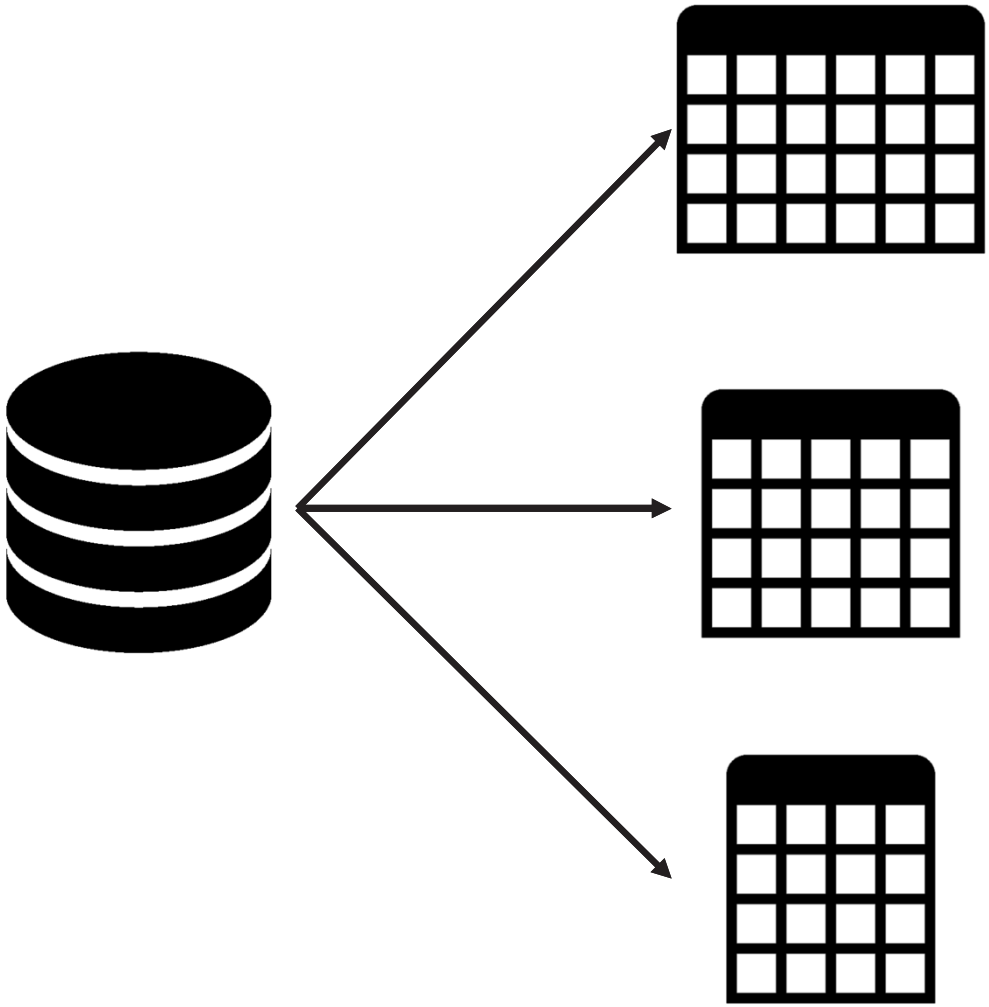
What is polymorphism

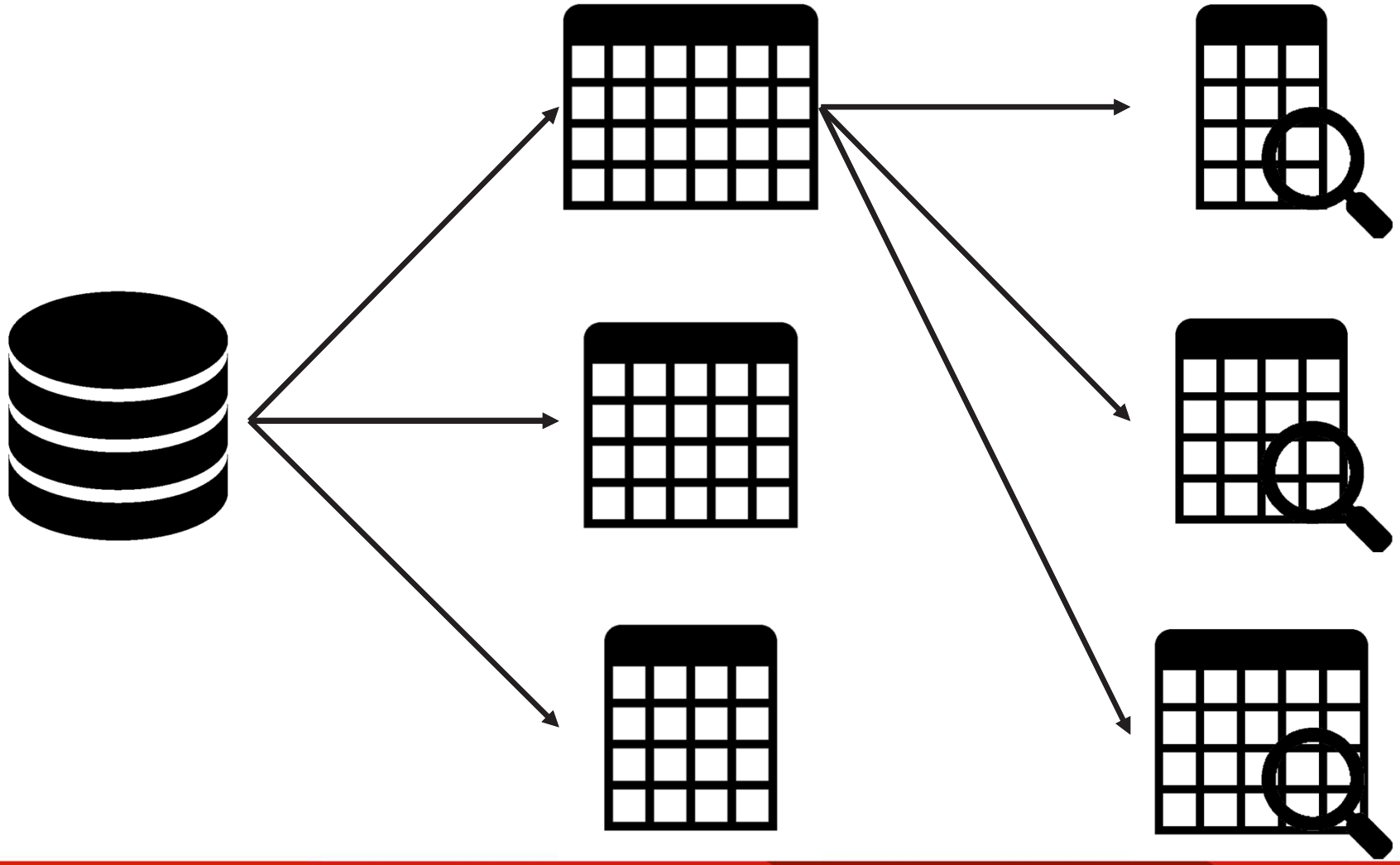
https://www.explore-share.com/trip/climb-mount-olympus-zagoria-hike/



https://www.haikudeck.com/greek-roots--uncategorized-presentation-nAYyrzqqim#slide12

https://www.haikudeck.com/greek-roots--uncategorized-presentation-nAYyrzqqim#slide8





Can you tell me in Tom and Jerry terms what POLYMORPHIC TABLE FUNCTIONS are?

What do they have in common

What do they have in common

What do they have in common

What do they have in common

Speak

Meow

Squeak

Woof

Static polymorphism

Static polymorphism

Polymorphism in PL/SQL (package)

```
create or replace package      pkg_TomAndJerry is
  procedure speak(cat_in   in varchar2);


  procedure speak(mouse_in in varchar2);


  procedure speak(dog_in   in varchar2);


end;
/
```

Polymorphism in PL/SQL (package)

```
create or replace package body pkg_TomAndJerry is
  procedure speak(cat_in   in varchar2) is
  begin
    dbms_output.put_line(cat_in || q'[ says 'Meow']');
  end;

  procedure speak(mouse_in in varchar2) is
  begin
    dbms_output.put_line(mouse_in || q'[ says 'Squeak']');
  end;

  procedure speak(dog_in   in varchar2) is
  begin
    dbms_output.put_line(dog_in || q'[ says 'Woof']');
  end;
end;
/
```

Polymorphism in PL/SQL (package)

```
begin
  dbms_output.put_line(q'[===Tom and Jerry sounds using package===]');
  pkg_TomAndJerry.speak(cat_in => 'Tom');
  pkg_TomAndJerry.speak(mouse_in => 'Jerry');
  pkg_TomAndJerry.speak(dog_in => 'Spike');
  dbms_output.put_line(q'[========================================]');
end;
/
```

```
===Tom and Jerry sounds using package===
Tom says 'Meow'
Jerry says 'Squeak'
Spike says 'Woof'
========================================
```

Dynamic polymorphism

Dynamic polymorphism

Polymorphism in PL/SQL (object type)

```
create or replace type TomAndJerry force as object
(
  name varchar2(100),
  member procedure speak
) not final
/
```

Polymorphism in PL/SQL (object type)

```
create or replace type TomAndJerry force as object
(
  name varchar2(100),
  member procedure speak
) not final
/

create or replace type Tom   under TomAndJerry
(
  overriding member procedure speak
)
/
create or replace type body Tom   is
  overriding member procedure speak is
  begin
    dbms_output.put_line(name || q'[ says 'Meow']'  );
  end;
end;
/
```

Polymorphism in PL/SQL (object type)

```
create or replace type TomAndJerry force as object
(
  name varchar2(100),
  member procedure speak
) not final
/

create or replace type Jerry under TomAndJerry
(
  overriding member procedure speak
)
/
create or replace type body Jerry is
  overriding member procedure speak is
  begin
    dbms_output.put_line(name || q'[ says 'Squeak']');
  end;
end;
/
```

Polymorphism in PL/SQL (object type)

```
create or replace type TomAndJerry force as object
(
  name varchar2(100),
  member procedure speak
) not final
/

create or replace type Spike under TomAndJerry
(
  overriding member procedure speak
)
/
create or replace type body Spike is
  overriding member procedure speak is
  begin
    dbms_output.put_line(name || q'[ says 'Woof']'  );
  end;
end;
/
```

Polymorphism in PL/SQL (object type)

```
declare
  l_TomAndJerry TomAndJerry;
begin
  dbms_output.put_line(q'[===Tom and Jerry sounds using types===]');
  l_TomAndJerry := Tom('Tom');
  l_TomAndJerry.speak;
  l_TomAndJerry := Jerry('Jerry');
  l_TomAndJerry.speak;
  l_TomAndJerry := Spike('Spike');
  l_TomAndJerry.speak;
  dbms_output.put_line(q'[=====================================]');
end;
/
```

```
===Tom and Jerry sounds using types===
Tom says 'Meow'
Jerry says 'Squeak'
Spike says 'Woof'
=====================================
```

Speak

Meow

Squeak

Woof

Speak

Meow

Squeak

Woof

Quack

Polymorphism in PL/SQL (object type)

```
create or replace type TomAndJerry force as object
(
  name varchar2(100),
  member procedure speak
) not final
/

create or replace type LittleQuacker under TomAndJerry
(
  overriding member procedure speak
)
/
create or replace type body LittleQuacker is
  overriding member procedure speak is
  begin
    dbms_output.put_line(name || q'[ says 'Quack']');
  end;
end;
/
```

Polymorphism in PL/SQL (object type)

```
declare
  l_TomAndJerry TomAndJerry;
begin
  dbms_output.put_line(q'[===Tom and Jerry sounds using types===]');
  l_TomAndJerry := Tom('Tom');
  l_TomAndJerry.speak;
  l_TomAndJerry := Jerry('Jerry');
  l_TomAndJerry.speak;
  l_TomAndJerry := Spike('Spike');
  l_TomAndJerry.speak;
  l_TomAndJerry := LittleQuacker('Little Quacker');
  l_TomAndJerry.speak;
  dbms_output.put_line(q'[=====================================]');
end;
```

```
===Tom and Jerry sounds using types===
Tom says 'Meow'
Jerry says 'Squeak'
Spike says 'Woof'
Little Quacker says 'Quack'
=====================================
```

Polymorphism using Table Functions

```
create table TomAndJerry
(name varchar2(100)
,sound varchar2(100)
)
/

insert into TomAndJerry (name, sound) values ('Tom','Meow')
/
insert into TomAndJerry (name, sound) values ('Jerry','Squeak’)
/
insert into TomAndJerry (name, sound) values ('Spike','Woof')
/
```

Polymorphism using Table Functions

```
create or replace type t_TomAndJerry force as object
(name varchar2(100)
,sound varchar2(100)
)
/

create or replace type tt_TomAndJerry as table of t_TomAndJerry
/
```

Polymorphism using Table Functions

```
create or replace package      pkg_TomAndJerry as
  function speak return tt_TomAndJerry
    pipelined;
```

```
end pkg_TomAndJerry;
/
```

Polymorphism using Table Functions

```
create or replace package body pkg_TomAndJerry as
  function speak return tt_TomAndJerry
    pipelined is
    cursor c_TomAndJerry is
      select t_TomAndJerry(name, sound) from TomAndJerry;
    l_returnvalue t_TomAndJerry;
  begin
    open c_TomAndJerry;
    loop
      fetch c_TomAndJerry
        into l_returnvalue;
      exit when c_TomAndJerry%notfound;
      pipe row(l_returnvalue);
    end loop;
    return;
  end;
end pkg_TomAndJerry;
/
```

Polymorphism using Table Functions

```
select name || ' says ' || sound
  from table(pkg_TomAndJerry.speak)
/
```

```
NAME||'SAYS'||SOUND
---------------------------
Spike says Woof
Tom says Meow
Jerry says Squeak
```

```
insert into TomAndJerry (name, sound) values ('Little Quacker','Quack')
/
```

```
select name || ' says ' || sound
  from table(pkg_TomAndJerry.speak)
/
```

```
NAME||'SAYS'||SOUND
---------------------------
Spike says Woof
Tom says Meow
Jerry says Squeak
Little Quacker says Quack
```

Speak

Meow

Squeak

Woof

Quack

Speak

Meow

Miauw

Squeak

Piep

Woof

Blaf

Quack

Kwek

Polymorphism using Table Functions

```
alter table TOMANDJERRY rename column sound to English
/
alter table TomAndJerry add (Dutch varchar2(100))
/

update TomAndJerry set Dutch = 'Blaf' where name = 'Spike'
/
update TomAndJerry set Dutch = 'Miauw' where name = 'Tom'
/
update TomAndJerry set Dutch = 'Piep' where name = 'Jerry'
/
update TomAndJerry set Dutch = 'Kwek' where name = 'Little Quacker'
/
```

Polymorphism using Table Functions

```
select name || ' says ' || sound
  from table(pkg_TomAndJerry.speak)
/
                         ORA-04063: package body "DEMO.PKG_TOMANDJERRY" has errors


create or replace type t_TomAndJerry force as object
(name varchar2(100)
,English varchar2(100)
,Dutch varchar2(100)
)
/
```

Polymorphism using Table Functions

```
select name || ' says ' || sound
  from table(pkg_TomAndJerry.speak)
/
```

```
ORA-00904: "SOUND": invalid identifier
```

```
select name || ' says ' || English
  from table(pkg_TomAndJerry.speak)
/
```

```
ORA-04063: package body "DEMO.PKG_TOMANDJERRY" has errors
```

Polymorphism using Table Functions

```
create or replace package body pkg_TomAndJerry as
  function speak return tt_TomAndJerry
    pipelined is
    cursor c_TomAndJerry is
      select t_TomAndJerry(name, English, Dutch) from TomAndJerry;
    l_returnvalue t_TomAndJerry;
  begin
    open c_TomAndJerry;
    loop
      fetch c_TomAndJerry
        into l_returnvalue;
      exit when c_TomAndJerry%notfound;
      pipe row(l_returnvalue);
    end loop;
    return;
  end;
end pkg_TomAndJerry;
/
```

Polymorphism using Table Functions

```
select name || ' says ' || English
  from table(pkg_TomAndJerry.speak)
/
```

```
NAME||'SAYS'||ENGLISH
-------------------------
Spike says Woof
Tom says Meow
Jerry says Squeak
Little Quacker says Quack
```

```
select name || ' says ' || Dutch
  from table(pkg_TomAndJerry.speak)
/
```

```
NAME||'SAYS'||DUTCH
-------------------------
Spike says Blaf
Tom says Miauw
Jerry says Piep
Little Quacker says Kwek
```

Speak

Meow

Miauw

Squeak

Piep

Woof

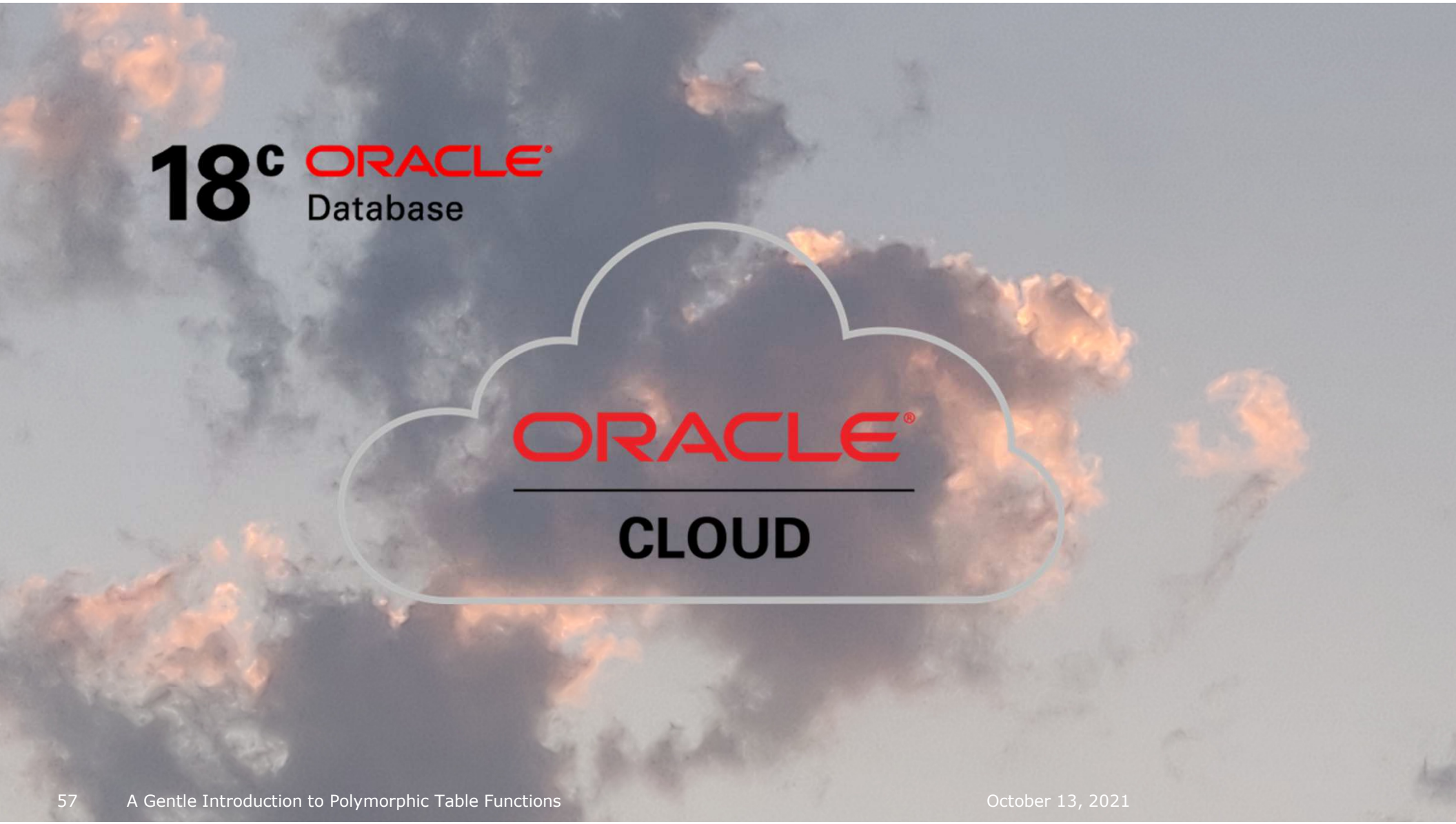
Blaf

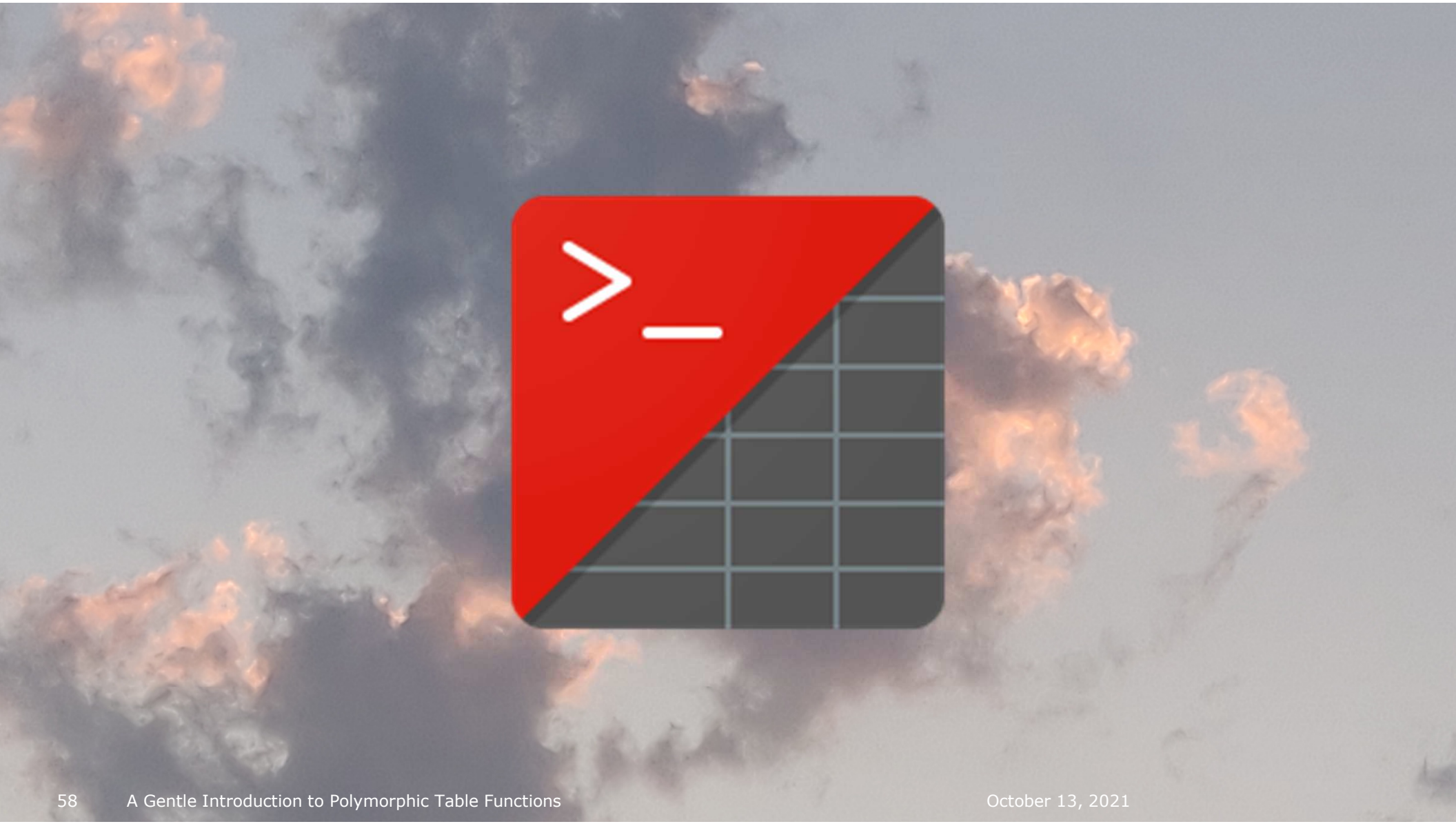
Quack

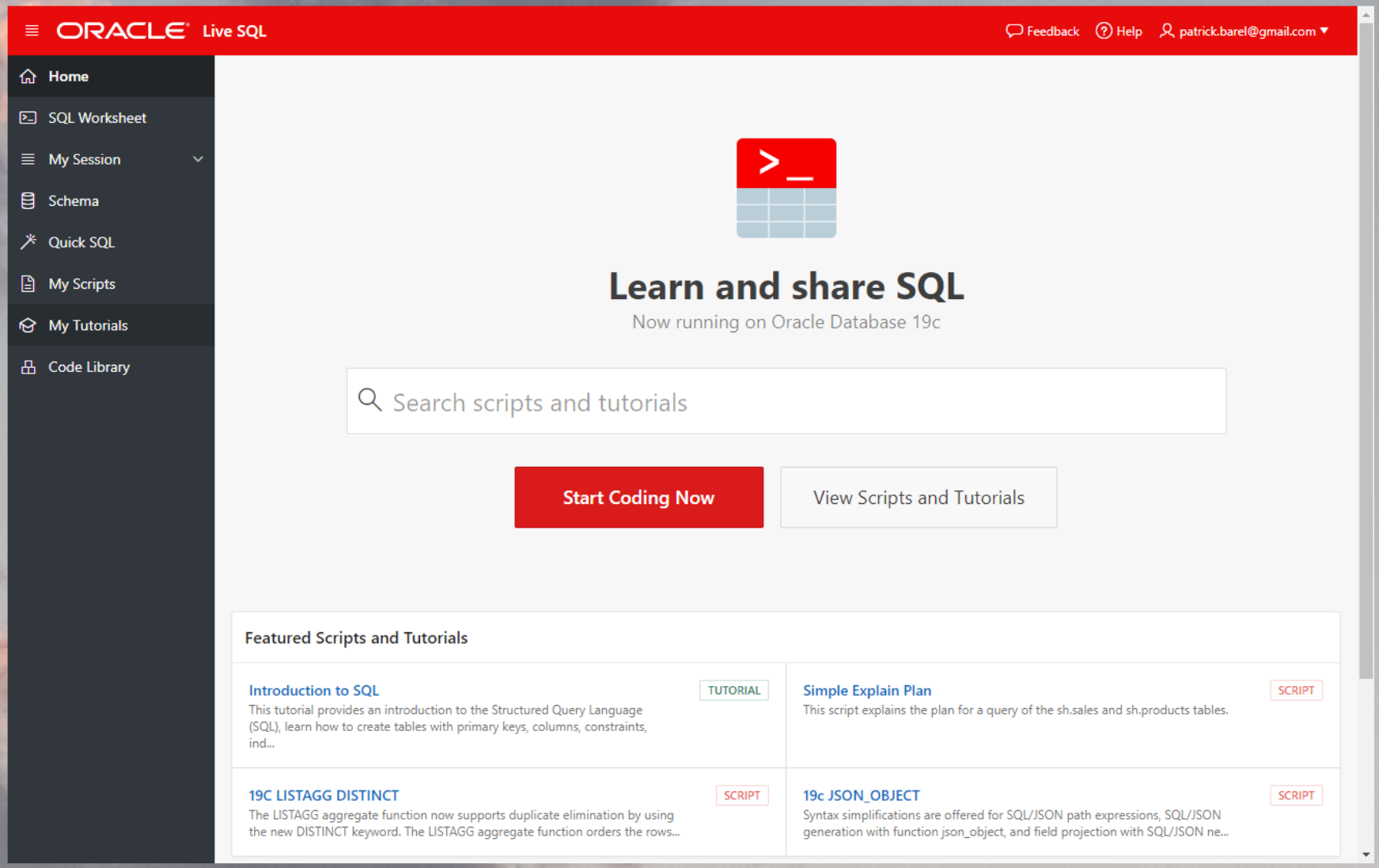
Kwek

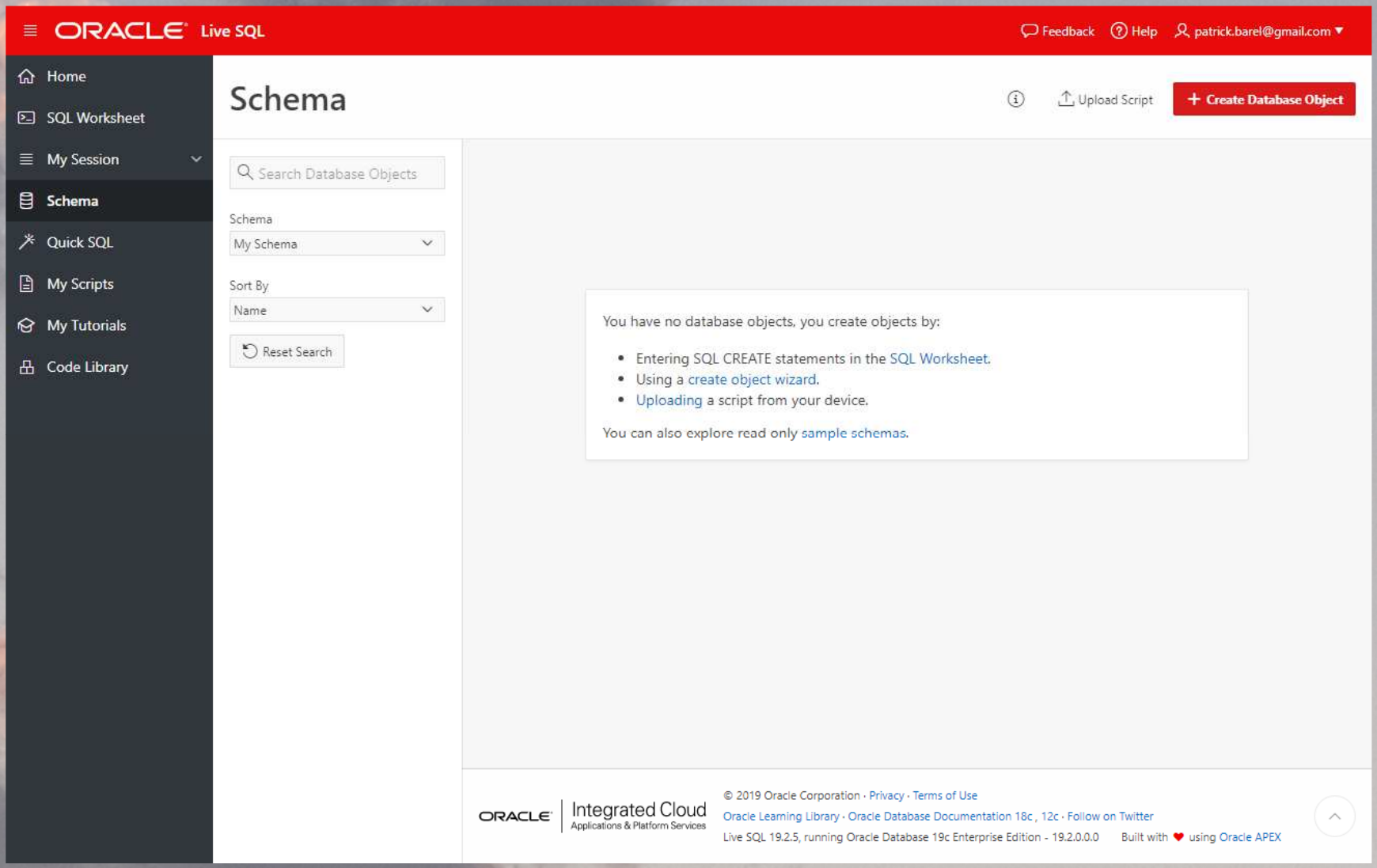
Speak

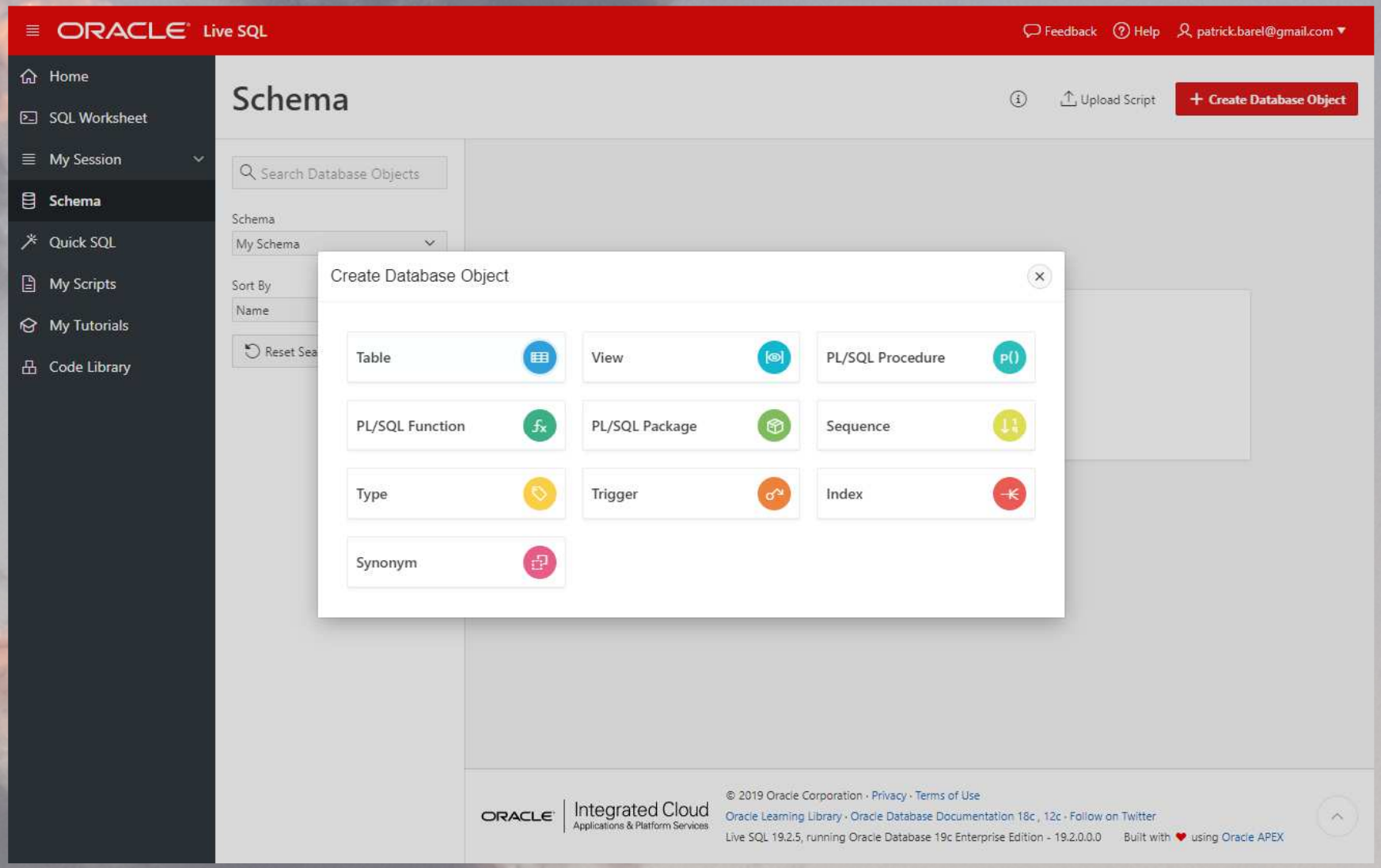
			
	Meow	Miauw	Miaou
	Squeak	Piep	Iiik
	Woof	Blaf	Guau
	Quack	Kwek	Cuac cuac

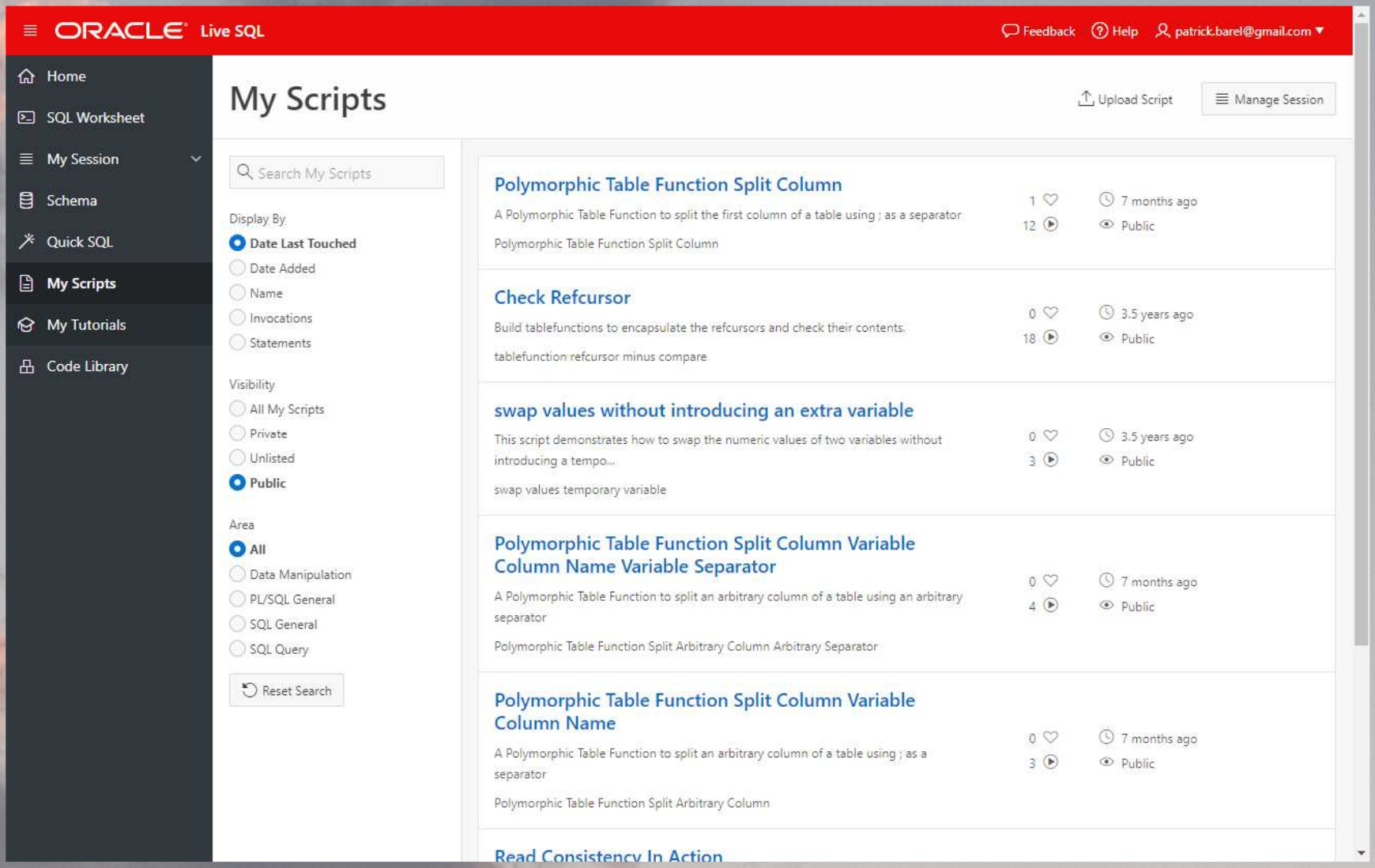

18c
ORACLE
Database
ORACLE
CLOUD

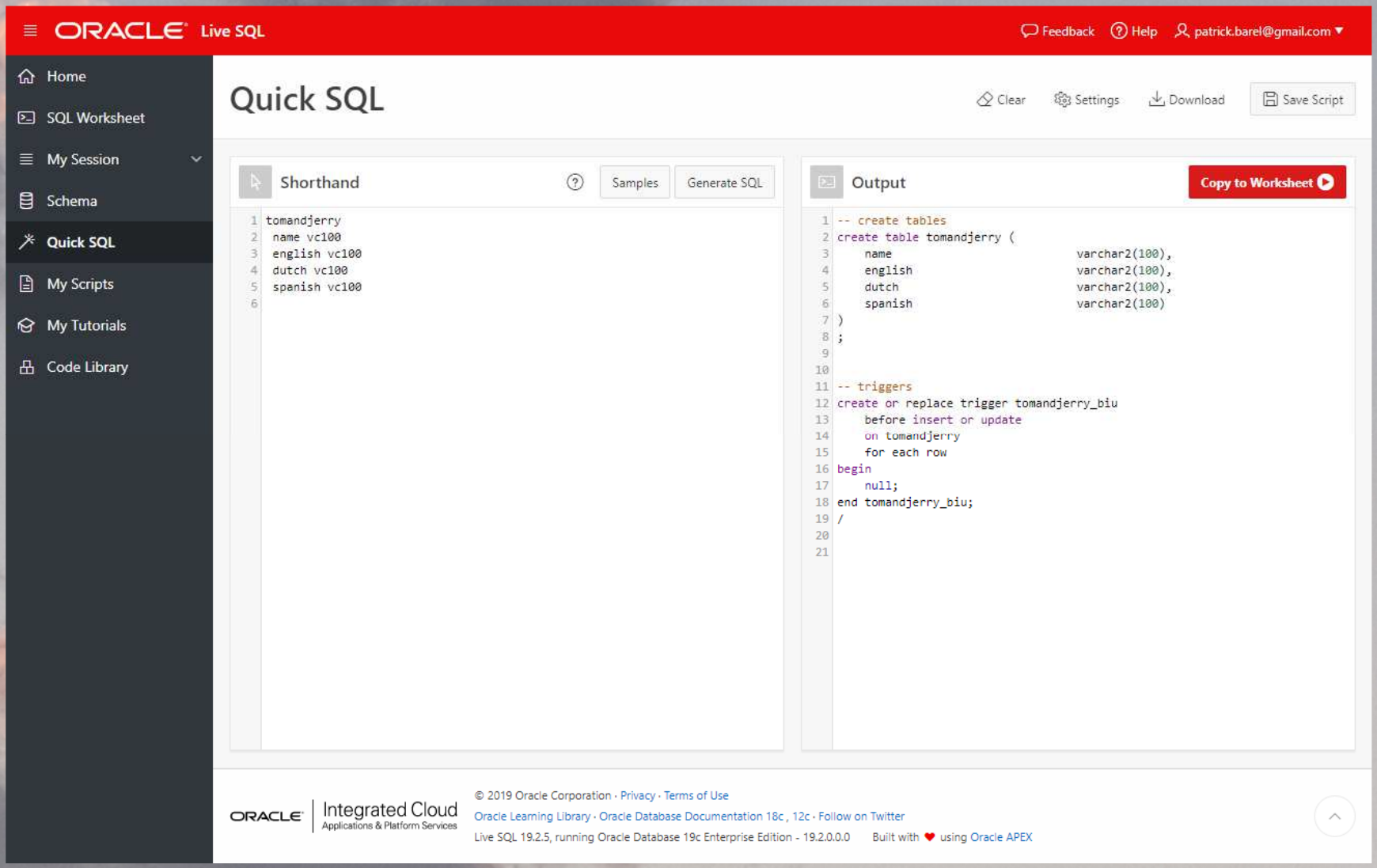



ORACLE Live SQL
Feedback
Help
patrick.barel@gmail.com
Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
My Tutorials
Code Library
Learn and share SQL
Now running on Oracle Database 19c
Search scripts and tutorials
Start Coding Now
View Scripts and Tutorials
Featured Scripts and Tutorials
Introduction to SQL
TUTORIAL
This tutorial provides an introduction to the Structured Query Language (SQL), learn how to create tables with primary keys, columns, constraints, ind...
Simple Explain Plan
SCRIPT
This script explains the plan for a query of the sh.sales and sh.products tables.
19C LISTAGG DISTINCT
SCRIPT
The LISTAGG aggregate function now supports duplicate elimination by using the new DISTINCT keyword. The LISTAGG aggregate function orders the rows...
19c JSON_OBJECT
SCRIPT
Syntax simplifications are offered for SQL/JSON path expressions, SQL/JSON generation with function json_object, and field projection with SQL/JSON ne...


ORACLE Live SQL
Feedback
Help
patrick.barel@gmail.com
Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
My Tutorials
Code Library
Schema
Upload Script
Create Database Object
Search Database Objects
Schema
My Schema
Sort By
Name
Reset Search
You have no database objects, you create objects by:
Entering SQL CREATE statements in the SQL Worksheet.
Using a create object wizard.
Uploading a script from your device.
You can also explore read only sample schemas.
ORACLE Integrated Cloud Applications & Platform Services
© 2019 Oracle Corporation · Privacy · Terms of Use
Oracle Learning Library · Oracle Database Documentation 18c , 12c · Follow on Twitter
Live SQL 19.2.5, running Oracle Database 19c Enterprise Edition - 19.2.0.0.0
Built with ♥ using Oracle APEX


ORACLE Live SQL
Feedback
Help
patrick.barel@gmail.com
Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
My Tutorials
Code Library
Schema
Upload Script
Create Database Object
Search Database Objects
Schema
My Schema
Sort By
Name
Reset Sea
Create Database Object
Table
View
PL/SQL Procedure
PL/SQL Function
PL/SQL Package
Sequence
Type
Trigger
Index
Synonym
ORACLE Integrated Cloud
Applications & Platform Services
© 2019 Oracle Corporation · Privacy · Terms of Use
Oracle Learning Library · Oracle Database Documentation 18c , 12c · Follow on Twitter
Live SQL 19.2.5, running Oracle Database 19c Enterprise Edition - 19.2.0.0.0
Built with ❤ using Oracle APEX


ORACLE Live SQL
Feedback
Help
patrick.barel@gmail.com
Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
My Tutorials
Code Library
My Scripts
Upload Script
Manage Session
Search My Scripts
Display By
Date Last Touched
Date Added
Name
Invocations
Statements
Visibility
All My Scripts
Private
Unlisted
Public
Area
All
Data Manipulation
PL/SQL General
SQL General
SQL Query
Reset Search
Polymorphic Table Function Split Column
A Polymorphic Table Function to split the first column of a table using ; as a separator
Polymorphic Table Function Split Column
1
12
7 months ago
Public
Check Refcursor
Build tablefunctions to encapsulate the refcursors and check their contents.
tablefunction refcursor minus compare
0
18
3.5 years ago
Public
swap values without introducing an extra variable
This script demonstrates how to swap the numeric values of two variables without introducing a tempo...
swap values temporary variable
0
3
3.5 years ago
Public
Polymorphic Table Function Split Column Variable Column Name Variable Separator
A Polymorphic Table Function to split an arbitrary column of a table using an arbitrary separator
Polymorphic Table Function Split Arbitrary Column Arbitrary Separator
0
4
7 months ago
Public
Polymorphic Table Function Split Column Variable Column Name
A Polymorphic Table Function to split an arbitrary column of a table using ; as a separator
Polymorphic Table Function Split Arbitrary Column
0
3
7 months ago
Public
Read Consistency In Action


ORACLE Live SQL
Feedback
Help
patrick.barel@gmail.com
Home
SQL Worksheet
My Session
Schema
Quick SQL
My Scripts
My Tutorials
Code Library
Quick SQL
Clear
Settings
Download
Save Script
Shorthand
Samples
Generate SQL
tomandjerry
name vc100
english vc100
dutch vc100
spanish vc100
Output
Copy to Worksheet
-- create tables
create table tomandjerry (
name varchar2(100),
english varchar2(100),
dutch varchar2(100),
spanish varchar2(100)
)
;
-- triggers
create or replace trigger tomandjerry_biu
before insert or update
on tomandjerry
for each row
begin
null;
end tomandjerry_biu;
/
ORACLE Integrated Cloud
Applications & Platform Services
© 2019 Oracle Corporation · Privacy · Terms of Use
Oracle Learning Library · Oracle Database Documentation 18c, 12c · Follow on Twitter
Live SQL 19.2.5, running Oracle Database 19c Enterprise Edition - 19.2.0.0.0
Built with ♥ using Oracle APEX

Oracle Cloud Infrastructure

New Free Tier

oracle.com/cloud/free

Always Free

Services you can use for unlimited time

+

30-Day Free Trial

Free credits you can use for more services

Polymorphic table functions

```
create table TomAndJerry
(name varchar2(100)
,English varchar2(100)
,Dutch varchar(100)
)
/

insert into TomAndJerry (name, English, Dutch) values ('Spike', 'Woof', 'Blaf')
/
insert into TomAndJerry (name, English, Dutch) values ('Tom', 'Meow', 'Miauw')
/
insert into TomAndJerry (name, English, Dutch) values ('Jerry', 'Squeak', 'Piep')
/
```

Polymorphic table functions

```
create or replace package pkg_TomAndJerry_ptf as

  function describe(tab in out dbms_tf.table_t, col dbms_tf.columns_t)
    return dbms_tf.describe_t;

end pkg_TomAndJerry_ptf;
/
```

Polymorphic table functions

```
create or replace package body pkg_TomAndJerry_ptf as
  function describe(tab in out dbms_tf.table_t, col dbms_tf.columns_t)
    return dbms_tf.describe_t as
    new_cols dbms_tf.columns_new_t;
    col_id   pls_integer := 1;
  begin
    for i in 1 .. tab.column.count() loop
      for j in 1 .. col.count() loop
        tab.column(i).pass_through := tab.column(i)
                                      .description.name = col(j);
        exit when tab.column(i).pass_through;
      end loop;
    end loop;
    return null;
  end;
end pkg_TomAndJerry_ptf;
/
```

Polymorphic table functions

```
create or replace function TomAndJerry_speak(tab table
                                             ,col columns)
       return table pipelined row polymorphic using pkg_TomAndJerry_ptf;
/
```

Polymorphism using Polymorphic Table Functions

```
select * from TomAndJerry_speak(TomAndJerry, COLUMNS(name, english))
/
```

NAME	ENGLISH
Spike	Woof
Tom	Meow
Jerry	Squeak

```
select * from TomAndJerry_speak(TomAndJerry, COLUMNS(name, english, dutch))
/
```

NAME	ENGLISH	DUTCH
Spike	Woof	Blaf
Tom	Meow	Miauw
Jerry	Squeak	Piep

Polymorphism using Polymorphic Table Functions

```
insert into TomAndJerry (name, English, Dutch) values ('Little Quacker', 'Quack', 'Kwek')
/
        select * from TomAndJerry_speak(TomAndJerry, COLUMNS(name, english))
        /
                              NAME                 ENGLISH
                              -------------------- --------------------
                              Spike                Woof
                              Tom                  Meow
                              Jerry                Squeak
                              Little Quacker       Quack

        select * from TomAndJerry_speak(TomAndJerry, COLUMNS(name, english, dutch))
        /
                              NAME                 ENGLISH              DUTCH
                              -------------------- -------------------- --------------------
                              Spike                Woof                 Blaf
                              Tom                  Meow                 Miauw
                              Jerry                Squeak               Piep
                              Little Quacker       Quack                Kwek
```

Polymorphism using Polymorphic Table Functions

```
alter table TomAndJerry add (Spanish varchar2(100))
/


update TomAndJerry set Spanish = 'Guau' where name = 'Spike';
/
update TomAndJerry set Spanish = 'Miau' where name = 'Tom'
/
update TomAndJerry set Spanish = 'Iiik' where name = 'Jerry'
/
update TomAndJerry set Spanish = 'Cuac cuac' where name = 'Little Quacker'
/
```

Polymorphism using Polymorphic Table Functions

```
select * from TomAndJerry_speak(TomAndJerry, COLUMNS(name, english))
/
 NAME                 ENGLISH
 -------------------- --------------------
 Spike                Woof
 Tom                  Meow
 Jerry                Squeak
 Little Quacker       Quack

select * from TomAndJerry_speak(TomAndJerry, COLUMNS(name, english, dutch, spanish))
/
 NAME                 ENGLISH              DUTCH                SPANISH
 -------------------- -------------------- -------------------- --------------------
 Spike                Woof                 Blaf                 Guau
 Tom                  Meow                 Miauw                Miau
 Jerry                Squeak               Piep                 Iiik
 Little Quacker       Quack                Kwek                 Cuac cuac
```

Polymorphism using Polymorphic Table Functions

```
create table numbers
(name number
,english varchar2(10)
,dutch varchar2(10)
,spanish varchar2(10)
)
/
insert into numbers (name, english, dutch, spanish) values (1, 'One', 'Een', 'Uno')
/
insert into numbers (name, english, dutch, spanish) values (2, 'Two', 'Twee', 'Dos')
/
insert into numbers (name, english, dutch, spanish) values (3, 'Three', 'Drie', 'Tres')
/
```

Polymorphism using Polymorphic Table Functions

```
select * from TomAndJerry_speak(numbers, COLUMNS(name, english, dutch, spanish))
/
                  NAME ENGLISH              DUTCH                SPANISH
 -------------------- -------------------- -------------------- --------------------
                     1 One                  Een                  Uno
                     2 Two                  Twee                 Dos
                     3 Three                Drie                 Tres


select * from TomAndJerry_speak(TomAndJerry, COLUMNS(name, english, dutch, spanish))
/
 NAME                 ENGLISH              DUTCH                SPANISH
 -------------------- -------------------- -------------------- --------------------
 Spike                Woof                 Blaf                 Guau
 Tom                  Meow                 Miauw                Miau
 Jerry                Squeak               Piep                 Iiik
 Little Quacker       Quack                Kwek                 Cuac cuac
```

Separated list




Tom/Jerry/Spike/Little Quacker

Separated List – Polymorphic Table Function

```
create or replace package      separated_ptf is
  function describe(tab  in out dbms_tf.table_t
                   ,cols in dbms_tf.columns_t default null) return dbms_tf.describe_t;

  procedure fetch_rows;
end separated_ptf;
/
```

Separated List – Polymorphic Table Function

```
create or replace package body separated_ptf as
  function describe(tab  in out dbms_tf.table_t
                   ,cols in dbms_tf.columns_t default null) return dbms_tf.describe_t as
    -- metadata for column to add
    l_new_col dbms_tf.column_metadata_t;
    -- table of columns to add
    l_new_cols dbms_tf.columns_new_t; -- := DBMS_TF.COLUMNS_NEW_T();
  begin
    -- Mark the first column ReadOnly and don't display it anymore
    tab.column(1).for_read := true;
    tab.column(1).pass_through := false;
    -- Add the new columns, as specified in the cols parameter
    for indx in 1 .. cols.count loop
      -- define metadata for column named cols(indx)
      -- that will default to a datatype of varchar2 with
      -- a length of 4000
      l_new_col := dbms_tf.column_metadata_t(name => cols(indx));
      -- add the new column to the list of columns new columns
```

Separated List – Polymorphic Table Function

```
create or replace package body separated_ptf as
  function describe(tab  in out dbms_tf.table_t
                   ,cols in dbms_tf.columns_t default null) return dbms_tf.describe_t as
    -- metadata for column to add
    l_new_col dbms_tf.column_metadata_t;
    -- table of columns to add
    l_new_cols dbms_tf.columns_new_t; -- := DBMS_TF.COLUMNS_NEW_T();
  begin
    -- Mark the first column ReadOnly and don't display it anymore
    tab.column(1).for_read := true;
    tab.column(1).pass_through := false;
    -- Add the new columns, as specified in the cols parameter
    for indx in 1 .. cols.count loop
      -- define metadata for column named cols(indx)
      -- that will default to a datatype of varchar2 with
      -- a length of 4000
      l_new_col := dbms_tf.column_metadata_t(name => cols(indx));
      -- add the new column to the list of columns new columns
```

Separated List – Polymorphic Table Function

```
  -- Add the new columns, as specified in the cols parameter
  for indx in 1 .. cols.count loop
    -- define metadata for column named cols(indx)
    -- that will default to a datatype of varchar2 with
    -- a length of 4000
    l_new_col := dbms_tf.column_metadata_t(name => cols(indx));
    -- add the new column to the list of columns new columns
    l_new_cols(l_new_cols.count + 1) := l_new_col;
  end loop;
  -- Instead of returning NULL we will RETURN a specific
  -- DESCRIBE_T that adds new columns
  return dbms_tf.describe_t(new_columns => l_new_cols);
end;

procedure fetch_rows is
  -- define a table type of varchar2 tables
  type colset is table of dbms_tf.tab_varchar2_t index by pls_integer;
  -- variable to hold the rowset as retrieved
  l_rowset dbms_tf.row_set_t;
  -- variable to hold the number of rows as retrieved
```

Separated List – Polymorphic Table Function

```
  -- Add the new columns, as specified in the cols parameter
  for indx in 1 .. cols.count loop
    -- define metadata for column named cols(indx)
    -- that will default to a datatype of varchar2 with
    -- a length of 4000
    l_new_col := dbms_tf.column_metadata_t(name => cols(indx));
    -- add the new column to the list of columns new columns
    l_new_cols(l_new_cols.count + 1) := l_new_col;
  end loop;
  -- Instead of returning NULL we will RETURN a specific
  -- DESCRIBE_T that adds new columns
  return dbms_tf.describe_t(new_columns => l_new_cols);
end;

procedure fetch_rows is
  -- define a table type of varchar2 tables
  type colset is table of dbms_tf.tab_varchar2_t index by pls_integer;
  -- variable to hold the rowset as retrieved
  l_rowset dbms_tf.row_set_t;
  -- variable to hold the number of rows as retrieved
```

Separated List – Polymorphic Table Function

```
end;

procedure fetch_rows is
  -- define a table type of varchar2 tables
  type colset is table of dbms_tf.tab_varchar2_t index by pls_integer;
  -- variable to hold the rowset as retrieved
  l_rowset dbms_tf.row_set_t;
  -- variable to hold the number of rows as retrieved
  l_rowcount pls_integer;
  -- variable to hold the number of put columns
  l_putcolcount pls_integer := dbms_tf.get_env().put_columns.count;
  -- variable to hold the new values
  l_newcolset colset;
  -- get the name of the column to be split from the get columns
  l_coltosplit dbms_quoted_id := trim('"' from dbms_tf.get_env().get_columns(1).name);
begin
  -- fetch rows into a local rowset
  -- at this point the rows will have columns
  -- from the table/view/query passed in
  dbms_tf.get_row_set(l_rowset, l_rowcount);
  -- for every row in the rowset
```

Separated List – Polymorphic Table Function

```
end;

procedure fetch_rows is
  -- define a table type of varchar2 tables
  type colset is table of dbms_tf.tab_varchar2_t index by pls_integer;
  -- variable to hold the rowset as retrieved
  l_rowset dbms_tf.row_set_t;
  -- variable to hold the number of rows as retrieved
  l_rowcount pls_integer;
  -- variable to hold the number of put columns
  l_putcolcount pls_integer := dbms_tf.get_env().put_columns.count;
  -- variable to hold the new values
  l_newcolset colset;
  -- get the name of the column to be split from the get columns
  l_coltosplit dbms_quoted_id := trim('"' from dbms_tf.get_env().get_columns(1).name);
begin
  -- fetch rows into a local rowset
  -- at this point the rows will have columns
  -- from the table/view/query passed in
  dbms_tf.get_row_set(l_rowset, l_rowcount);
  -- for every row in the rowset
```

Separated List – Polymorphic Table Function

```
  -- get the name of the column to be split from the get columns
  l_coltosplit dbms_quoted_id := trim('"' from dbms_tf.get_env().get_columns(1).name);
begin
  -- fetch rows into a local rowset
  -- at this point the rows will have columns
  -- from the table/view/query passed in
  dbms_tf.get_row_set(l_rowset, l_rowcount);
  -- for every row in the rowset...
  for rowindx in 1 .. l_rowcount loop
    -- for every column
    for colindx in 1 .. l_putcolcount loop
      -- split the row into separate values
      l_newcolset(colindx)(rowindx) := trim(',' from
        regexp_substr(json_value(dbms_tf.row_to_char(l_rowset, rowindx),'$.' || l_coltosplit)
                      , '[^,]*,{0,1}'
                      , 1
                      , colindx));
    end loop; -- every column
  end loop; -- every row in the rowset
  -- add the newly populated columns to the rowset
```

Separated List – Polymorphic Table Function

```
  -- get the name of the column to be split from the get columns
  l_coltosplit dbms_quoted_id := trim('"' from dbms_tf.get_env().get_columns(1).name);
begin
  -- fetch rows into a local rowset
  -- at this point the rows will have columns
  -- from the table/view/query passed in
  dbms_tf.get_row_set(l_rowset, l_rowcount);
  -- for every row in the rowset...
  for rowindx in 1 .. l_rowcount loop
    -- for every column
    for colindx in 1 .. l_putcolcount loop
      -- split the row into separate values
      l_newcolset(colindx)(rowindx) := trim(',' from
        regexp_substr(json_value(dbms_tf.row_to_char(l_rowset, rowindx),'$.' || l_coltosplit)
                      , '[^,]*,{0,1}'
                      , 1
                      , colindx));
    end loop; -- every column
  end loop; -- every row in the rowset
  -- add the newly populated columns to the rowset
```

Separated List – Polymorphic Table Function

```
        l_newcolset(colindx)(rowindx) := trim(',' from
          regexp_substr(json_value(dbms_tf.row_to_char(l_rowset, rowindx),'$.' || l_coltosplit)
                        , '[^,]*,{0,1}'
                        , 1
                        , colindx));
      end loop; -- every column
    end loop; -- every row in the rowset
    -- add the newly populated columns to the rowset
    for indx in 1 .. l_putcolcount loop
      dbms_tf.put_col(columnid => indx, collection => l_newcolset(indx));
    end loop;
  end;
end separated_ptf;
/
```

Separated List – Polymorphic Table Function

```
        l_newcolset(colindx)(rowindx) := trim(',' from
          regexp_substr(json_value(dbms_tf.row_to_char(l_rowset, rowindx),'$.' || l_coltosplit)
                        , '[^,]*,{0,1}'
                        , 1
                        , colindx));
      end loop; -- every column
    end loop; -- every row in the rowset
    -- add the newly populated columns to the rowset
    for indx in 1 .. l_putcolcount loop
      dbms_tf.put_col(columnid => indx, collection => l_newcolset(indx));
    end loop;
  end;
end separated_ptf;
/
```

Separated List – Polymorphic Table Function

```
-- create a 'wrapper' function for the polymorphic table function
create or replace function separated_fnc(p_tbl      in table
                                        ,cols       columns default null) return table
  pipelined row polymorphic using separated_ptf;
/
```

Separated list – Tradional SQL

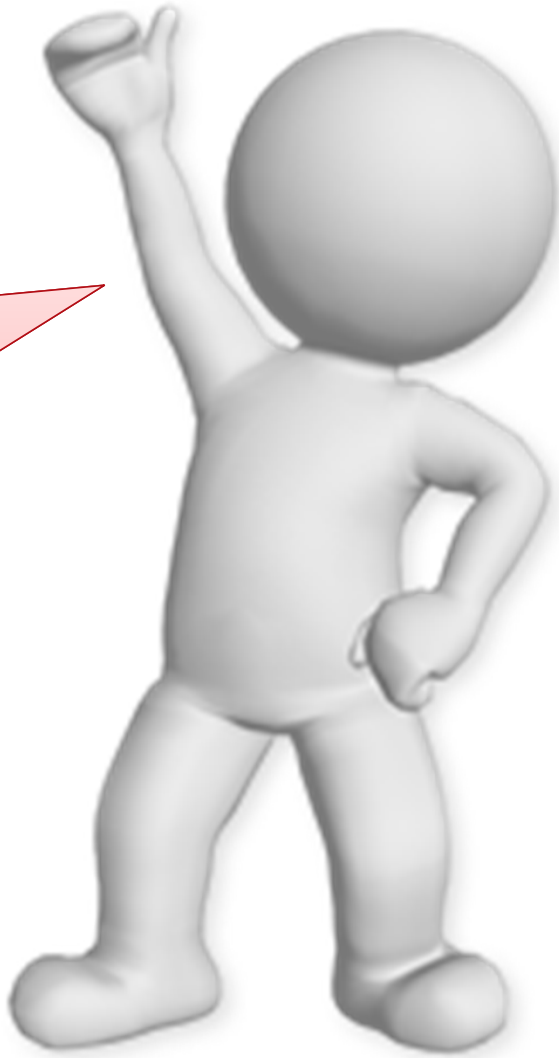
```
with tomandjerry as
 (select 'Tom,Jerry,Spike,Little Quacker' names
    from dual)
select trim(regexp_substr(names, '[^,]+', 1, 1)) first
      ,trim(regexp_substr(names, '[^,]+', 1, 2)) second
      ,trim(regexp_substr(names, '[^,]+', 1, 3)) third
      ,trim(regexp_substr(names, '[^,]+', 1, 4)) fourth
  from tomandjerry
/

FIRST SECOND THIRD FOURTH
----- ------ ----- --------------
Tom   Jerry  Spike Little Quacker
```

Separated list – Polymorphic Table Function

```
with tomandjerry as
 (select 'Tom,Jerry,Spike,Little Quacker' names
    from dual)
select *
  from separated_fnc(tomandjerry, columns(first, second, third, fourth))
/
```

```
FIRST SECOND THIRD FOURTH
----- ------ ----- --------------
Tom   Jerry  Spike Little Quacker
```

Hey
You are looking for use-cases for **Polymorphic Table Functions**?
Why don't you build a sum(interval) function?



Hmmm…
That will be a nice use case for a
Table Semantics
Polymorphic Table Function

SUM(INTERVAL)

```
prompt create the intervals table
create table intervals
( id number
, intervalym interval year to month
, intervalds interval day to second
)
/
create the intervals table

Table created
```

SUM(INTERVAL)

```
prompt add some data to the table
begin
  insert into intervals(id, intervalym, intervalds)
  values (1, INTERVAL '1-1' YEAR TO MONTH, INTERVAL '1 00:00:01.001' DAY TO SECOND(3));
  insert into intervals(id, intervalym, intervalds)
  values (2, INTERVAL '2-2' YEAR TO MONTH, INTERVAL '2 00:00:02.002' DAY TO SECOND(3));
  insert into intervals(id, intervalym, intervalds)
  values (3, INTERVAL '3-3' YEAR TO MONTH, INTERVAL '3 00:00:03.003' DAY TO SECOND(3));
  insert into intervals(id, intervalym, intervalds)
  values (4, INTERVAL '4-4' YEAR TO MONTH, INTERVAL '4 00:00:04.004' DAY TO SECOND(3));
  commit;
end;
/
add some data to the table

PL/SQL procedure successfully completed
```

SUM(INTERVAL)

```
prompt show the data
select * from intervals
/
show the data

        ID INTERVALYM           INTERVALDS
---------- -------------------- --------------------
         1 +01-01               +01 00:00:01.001000
         2 +02-02               +02 00:00:02.002000
         3 +03-03               +03 00:00:03.003000
         4 +04-04               +04 00:00:04.004000
```

SUM(INTERVAL)

```
prompt try to get a sum of the YEAR TO MONTH INTERVALs
select sum(intervalym) from intervals
/
try to get a sum of the YEAR TO MONTH INTERVALs
select sum(intervalym) from intervals

ORA-00932: inconsistent datatypes: expected NUMBER got INTERVAL YEAR TO MONTH
prompt it can only be done by adding the individual values
select INTERVAL '1-1' YEAR TO MONTH +
       INTERVAL '2-2' YEAR TO MONTH +
       INTERVAL '3-3' YEAR TO MONTH +
       INTERVAL '4-4' YEAR TO MONTH
  from dual
/
it can only be done by adding the individual values

INTERVAL'1-1'YEARTOMONTH+INTERVAL'2-2'YEARTOMONTH+INTERVAL'3-3'YEARTOMONTH+INTER
--------------------------------------------------------------------------------
+000000010-10
```

SUM(INTERVAL)

```
prompt try to get a sum of the DAY TO SECOND INTERVALs
select sum(intervalds) from intervals
/
try to get a sum of the DAY TO SECOND INTERVALs
select sum(intervalds) from intervals

ORA-00932: inconsistent datatypes: expected NUMBER got INTERVAL DAY TO SECOND
prompt it can only be done by adding the individual values
select INTERVAL '1 00:00:01.001' DAY TO SECOND(3) +
       INTERVAL '2 00:00:02.002' DAY TO SECOND(3) +
       INTERVAL '3 00:00:03.003' DAY TO SECOND(3) +
       INTERVAL '4 00:00:04.004' DAY TO SECOND(3)
  from dual
/
it can only be done by adding the individual values

INTERVAL'100:00:01.001'DAYTOSECOND(3)+INTERVAL'200:00:02.002'DAYTOSECOND(3)+INTE
--------------------------------------------------------------------------------
+000000010 00:00:10.010000000
```

SUM(INTERVAL)

```
create or replace package suminterval_ptf is
  function describe(tab              in out dbms_tf.table_t
                   ,cols         in dbms_tf.columns_t default null
                   ) return dbms_tf.describe_t;

  procedure fetch_rows;
end suminterval_ptf;
/
```

SUM(INTERVAL)

```
create or replace package body suminterval_ptf as
  -- Record type to hold the different INTERVAL sums
  type sum_rec is record(
     sumym interval year to month
    ,sumds interval day to second);
  -- Collection type for every column
  type sum_recs is table of sum_rec index by pls_integer;
  --
  function describe(tab  in out dbms_tf.table_t
                   ,cols in dbms_tf.columns_t default null)
  return dbms_tf.describe_t as
    sum_cols dbms_tf.columns_new_t;
  begin
    -- check every column from the source table
    for indx in tab.column.first .. tab.column.last loop
      -- mark every columns pass_through as false
      -- so it won't show up in the result anymore
      tab.column(indx).pass_through := false;
      -- first mark the column not to be read, unless...
```

SUM(INTERVAL)

```
                    ,cols in dbms_tf.columns_t default null)
  return dbms_tf.describe_t as
    sum_cols dbms_tf.columns_new_t;
  begin
    -- check every column from the source table
    for indx in tab.column.first .. tab.column.last loop
      -- mark every columns pass_through as false
      -- so it won't show up in the result anymore
      tab.column(indx).pass_through := false;
      -- first mark the column not to be read, unless...
      tab.column(indx).for_read := false;
      for colindx in cols.first .. cols.last loop
        if tab.column(indx).description.name = cols(colindx) then
          -- ...the result of the sum is requested
          -- then read this column
          tab.column(indx).for_read := true;
          -- and add a new column of the same type but with the name SUM_colname_
          sum_cols(colindx) :=
            dbms_tf.column_metadata_t
                (name => 'SUM_' || replace(tab.column(indx).description.name,'"') || '_'
```

SUM(INTERVAL)

```
    tab.column(indx).for_read := false;
    for colindx in cols.first .. cols.last loop
      if tab.column(indx).description.name = cols(colindx) then
        -- ...the result of the sum is requested
        -- then read this column
        tab.column(indx).for_read := true;
        -- and add a new column of the same type but with the name SUM_colname_
        sum_cols(colindx) :=
          dbms_tf.column_metadata_t
               (name => 'SUM_' || replace(tab.column(indx).description.name,'"') || '_'
               ,type => tab.column(indx).description.type);
      end if;
    end loop;
  end loop;
  -- Instead of returning NULL we will RETURN a specific
  -- DESCRIBE_T that adds new columns
  return dbms_tf.describe_t(new_columns => sum_cols);
end;

procedure fetch_rows is
```

SUM(INTERVAL)

```
                ,type => tab.column(indx).description.type);
      end if;
    end loop;
  end loop;
  -- Instead of returning NULL we will RETURN a specific
  -- DESCRIBE_T that adds new columns
  return dbms_tf.describe_t(new_columns => sum_cols);
end;

procedure fetch_rows is
  -- variable to hold the rowset as retrieved
  l_rowset dbms_tf.row_set_t;
  -- variable to hold the number of rows as retrieved
  l_rowcount pls_integer;
  -- variable to hold the sum of each column
  l_sum_recs sum_recs;
  -- variable to hold the enviroment value
  env dbms_tf.env_t := dbms_tf.get_env();
  -- variable to hold all the YEAR TO MONTH INTERVALs
  l_intervalym dbms_tf.tab_interval_ym_t;
```

SUM(INTERVAL)

```
  -- variable to hold the rowset as retrieved
  l_rowset dbms_tf.row_set_t;
  -- variable to hold the number of rows as retrieved
  l_rowcount pls_integer;
  -- variable to hold the sum of each column
  l_sum_recs sum_recs;
  -- variable to hold the enviroment value
  env dbms_tf.env_t := dbms_tf.get_env();
  -- variable to hold all the YEAR TO MONTH INTERVALs
  l_intervalym dbms_tf.tab_interval_ym_t;
  -- variable to hold all the DAY TO SECOND INTERVALs
  l_intervalds dbms_tf.tab_interval_ds_t;
begin
  for colindx in 1 .. env.get_columns.count loop
    case env.get_columns(colindx).type
    -- when the column type is INTERVAL YEAR TO MONTH
      when dbms_tf.type_interval_ym then
        -- Get the contents of the column
        dbms_tf.get_col(columnid => colindx, collection => l_intervalym);
        -- Initialize the record value,
```

SUM(INTERVAL)

```
  -- variable to hold all the DAY TO SECOND INTERVALs
  l_intervalds dbms_tf.tab_interval_ds_t;
begin
  for colindx in 1 .. env.get_columns.count loop
    case env.get_columns(colindx).type
    -- when the column type is INTERVAL YEAR TO MONTH
      when dbms_tf.type_interval_ym then
        -- Get the contents of the column
        dbms_tf.get_col(columnid => colindx, collection => l_intervalym);
        -- Initialize the record value,
        -- otherwise you'll add something to NULL which results in NULL
        l_sum_recs(colindx).sumym := interval '0-0' year to month;
        -- Loop through all the values and add them together
        for indx in 1 .. l_intervalym.count loop
          l_sum_recs(colindx).sumym := l_sum_recs(colindx)
                                       .sumym + l_intervalym(indx);
        end loop;
        -- when the column type is INTERVAL DAY TO SECOND
      when dbms_tf.type_interval_ds then
        -- Get the contents of the column
```

SUM(INTERVAL)

```
    -- otherwise you'll add something to NULL which results in NULL
    l_sum_recs(colindx).sumym := interval '0-0' year to month;
    -- Loop through all the values and add them together
    for indx in 1 .. l_intervalym.count loop
      l_sum_recs(colindx).sumym := l_sum_recs(colindx)
                                   .sumym + l_intervalym(indx);
    end loop;
    -- when the column type is INTERVAL DAY TO SECOND
  when dbms_tf.type_interval_ds then
    -- Get the contents of the column
    dbms_tf.get_col(columnid => colindx, collection => l_intervalds);
    -- Initialize the record value,
    -- otherwise you'll add something to NULL which results in NULL
    l_sum_recs(colindx).sumds := interval '0 0:0:0' day to second;
    -- Loop through all the values and add them together
    for indx in 1 .. l_intervalds.count loop
      l_sum_recs(colindx).sumds := l_sum_recs(colindx)
                                   .sumds + l_intervalds(indx);
    end loop;
  else
```

SUM(INTERVAL)

```
        dbms_tf.get_col(columnid => colindx, collection => l_intervalds);
        -- Initialize the record value,
        -- otherwise you'll add something to NULL which results in NULL
        l_sum_recs(colindx).sumds := interval '0 0:0:0' day to second;
        -- Loop through all the values and add them together
        for indx in 1 .. l_intervalds.count loop
          l_sum_recs(colindx).sumds := l_sum_recs(colindx)
                                       .sumds + l_intervalds(indx);
        end loop;
      else
        -- Catch all others
        dbms_output.put_line(q'[Columns of this type (]' ||
                             env.get_columns(colindx).type ||
                             q'[) are not supported (yet).]');
    end case;
  end loop;
  -- completely ignore the current rowset from now on,
  -- just start a new set, with just the totals
  -- loop through the put_columns to fill the resulting row
  for colindx in 1 .. env.put_columns.count loop
```

SUM(INTERVAL)

```
      -- Catch all others
      dbms_output.put_line(q'[Columns of this type (]' ||
                           env.get_columns(colindx).type ||
                           q'[) are not supported (yet).]');
  end case;
end loop;
-- completely ignore the current rowset from now on,
-- just start a new set, with just the totals
-- loop through the put_columns to fill the resulting row
for colindx in 1 .. env.put_columns.count loop
  case env.put_columns(colindx).type
  -- when the column type is INTERVAL YEAR TO MONTH
    when dbms_tf.type_interval_ym then
      -- add this value to the resulting row
      l_rowset(colindx).tab_interval_ym(1) := l_sum_recs(colindx).sumym;
      -- when the column type is INTERVAL DAY TO SECOND
    when dbms_tf.type_interval_ds then
      -- add this value to the resulting row
      l_rowset(colindx).tab_interval_ds(1) := l_sum_recs(colindx).sumds;
  end case;
```

SUM(INTERVAL)

```
      case env.put_columns(colindx).type
      -- when the column type is INTERVAL YEAR TO MONTH
        when dbms_tf.type_interval_ym then
          -- add this value to the resulting row
          l_rowset(colindx).tab_interval_ym(1) := l_sum_recs(colindx).sumym;
          -- when the column type is INTERVAL DAY TO SECOND
        when dbms_tf.type_interval_ds then
          -- add this value to the resulting row
          l_rowset(colindx).tab_interval_ds(1) := l_sum_recs(colindx).sumds;
      end case;
    end loop;
    dbms_tf.put_row_set(l_rowset);
  end;
end suminterval_ptf;
/
sho err
```

SUM(INTERVAL)

```
    end loop;
    dbms_tf.put_row_set(l_rowset);
  end;
end suminterval_ptf;
/
sho err
```

SUM(INTERVAL)

```
-- create a 'wrapper' function for the polymorphic table function
create or replace function suminterval_fnc(p_tbl           in table
                                          ,cols       columns default null) return table
  pipelined table polymorphic using suminterval_ptf;
/
sho err
```

SUM(INTERVAL)

```
prompt try to get a sum of the YEAR TO MONTH INTERVALs
select sum(intervalym) from intervals
/
try to get a sum of the YEAR TO MONTH INTERVALs
select sum(intervalym) from intervals

ORA-00932: inconsistent datatypes: expected NUMBER got INTERVAL YEAR TO MONTH


prompt select the SUM of the YEAR TO MONTH INTERVALs using the Polymorphic Table Function
select *
  from suminterval_fnc(intervals, columns(intervalym))
/
select the SUM of the YEAR TO MONTH INTERVALs using the Polymorphic Table Function

SUM_INTERVALYM_
-------------------------------------------------------------------------------
+10-10
```

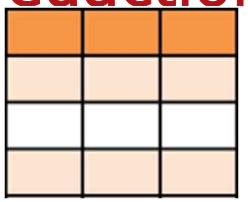
SUM(INTERVAL)

```
prompt try to get a sum of the DAY TO SECOND INTERVALs
select sum(intervalds) from intervals
/
try to get a sum of the DAY TO SECOND INTERVALs
select sum(intervalds) from intervals

ORA-00932: inconsistent datatypes: expected NUMBER got INTERVAL DAY TO SECOND


prompt select the SUM of the DAY TO SECONDS INTERVALs using the Polymorphic Table Function
select *
  from suminterval_fnc(intervals, columns(intervalds))
/
select the SUM of the DAY TO SECONDS INTERVALs using the Polymorphic Table Function

SUM_INTERVALDS_
---------------------------------------------------------------------------
+10 00:00:10
```

Data redaction

ContractData (Clear)	ContractData (Redacted)
9743-75970162-9280	XXXX-XXXXXXXX-9280
5699-91138137-7279	XXXX-XXXXXXXX-7279
1354-91983238-3840	XXXX-XXXXXXXX-3840

Data redaction



```
create table ContractData (contract varchar2(32))

insert into ContractData(contract) values ('9743-75970162-9280');
insert into ContractData(contract) values ('5699-91138137-7279');
insert into ContractData(contract) values ('1354-91983238-3840');
```

Data redaction

```
begin
  dbms_redact.add_policy(
    object_schema        => 'DEMO'
   ,object_name          => 'SENSITIVEDATA'
   ,policy_name          => 'PARTIALMASKDEMO'
   ,column_name          => 'CONTRACT'
   ,function_type        => DBMS_REDACT.PARTIAL
   ,function_parameters =>
      'VVVVFVVVVVVVVFVVVV, VVVV-VVVVVVVV-VVVV,X,1,12'
   ,expression           =>
      q'[SYS_CONTEXT ('USERENV', 'SESSION_USER') <> 'BU']');
end;
```

Data redaction

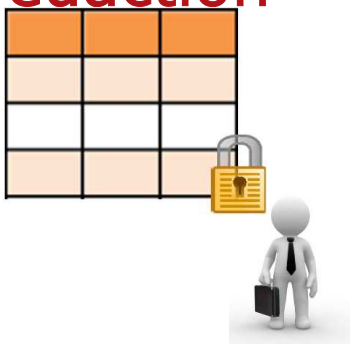
Different redaction types available

DBMS_REDACT.NONE
DBMS_REDACT.FULL
DBMS_REDACT.PARTIAL
DBMS_REDACT.RANDOM

http://docs.oracle.com/database/121/ARPLS/d_redact.htm

```
begin
  dbms_redact.add_policy(
    object_schema        => 'DEMO'
   ,object_name          => 'SENSITIVEDATA'
   ,policy_name          => 'PARTIALMASKDEMO'
   ,column_name          => 'CONTRACT'
   ,function_type        => DBMS_REDACT.PARTIAL
   ,function_parameters =>
      'VVVVFVVVVVVVVFVVVV, VVVV-VVVVVVVV-VVVV,X,1,12'
   ,expression           =>
      q'[SYS_CONTEXT ('USERENV', 'SESSION_USER') <> 'BU']');
end;
```

Data redaction



```
select * from demo.ContractData

CONTRACT
-----------------------
9743-75970162-9280
5699-91138137-7279
1354-91983238-3840
```

```
select * from demo.ContractData

CONTRACT
-----------------------
XXXX-XXXXXXXX-9280
XXXX-XXXXXXXX-7279
XXXX-XXXXXXXX-3840
```

Poor man's redaction

Data redaction

```
create or replace package redaction_ptf is
  function describe(tab           in out dbms_tf.table_t
                   ,cols          in dbms_tf.columns_t default null
                   ,redact_first in number default 0) return dbms_tf.describe_t;

  procedure fetch_rows(redact_first in number default 0);
end redaction_ptf;
/
```

Data redaction

```
create or replace package body redaction_ptf as
  subtype maxvarchar2 is varchar2(32767);
  function describe(tab          in out dbms_tf.table_t
                   ,cols         in dbms_tf.columns_t default null
                   ,redact_first in number default 0) return dbms_tf.describe_t as
    -- metadata for column to add
    l_new_col dbms_tf.column_metadata_t;
    -- table of columns to add
    l_new_cols dbms_tf.columns_new_t; -- := DBMS_TF.COLUMNS_NEW_T();
  begin
    -- find the columns to redact
    -- mark them for_read, but not pass_through
    <<outer_loop>>
    for indx in tab.column.first .. tab.column.last loop
      <<inner_loop>>
      for idx in 1 .. cols.count loop
        if tab.column(indx).description.name = cols(idx) then
          tab.column(indx).for_read := true;
          tab.column(indx).pass_through := false;
```

Data redaction

```
begin
  -- find the columns to redact
  -- mark them for_read, but not pass_through
  <<outer_loop>>
  for indx in tab.column.first .. tab.column.last loop
    <<inner_loop>>
    for idx in 1 .. cols.count loop
      if tab.column(indx).description.name = cols(idx) then
        tab.column(indx).for_read := true;
        tab.column(indx).pass_through := false;
        -- define metadata for column named cols(indx)
        -- that will default to a datatype of varchar2 with
        -- a length of 4000
        l_new_col := dbms_tf.column_metadata_t(name => tab.column(indx).description.name);
        -- add the new column to the list of columns new columns
        l_new_cols(l_new_cols.count + 1) := l_new_col;
        -- we are done, so we can exit this loop and go to the next column
        exit inner_loop;
      end if;
    end loop;
```

Data redaction

```
        -- define metadata for column named cols(indx)
        -- that will default to a datatype of varchar2 with
        -- a length of 4000
        l_new_col := dbms_tf.column_metadata_t(name => tab.column(indx).description.name);
        -- add the new column to the list of columns new columns
        l_new_cols(l_new_cols.count + 1) := l_new_col;
        -- we are done, so we can exit this loop and go to the next column
        exit inner_loop;
      end if;
    end loop;
  end loop;
  -- Instead of returning NULL we will RETURN a specific
  -- DESCRIBE_T that adds new columns
  return dbms_tf.describe_t(new_columns => l_new_cols);
end;

procedure fetch_rows(redact_first in number default 0) is
  -- define a table type of varchar2 tables
  type colset is table of dbms_tf.tab_varchar2_t index by pls_integer;
  -- variable to hold the rowset as retrieved
```

Data redaction

```
  end loop;
  -- Instead of returning NULL we will RETURN a specific
  -- DESCRIBE_T that adds new columns
  return dbms_tf.describe_t(new_columns => l_new_cols);
end;

procedure fetch_rows(redact_first in number default 0) is
  -- define a table type of varchar2 tables
  type colset is table of dbms_tf.tab_varchar2_t index by pls_integer;
  -- variable to hold the rowset as retrieved
  l_rowset dbms_tf.row_set_t;
  -- variable to hold the number of rows as retrieved
  l_rowcount pls_integer;
  -- variable to hold the number of put columns
  l_putcolcount pls_integer := dbms_tf.get_env().put_columns.count;
  -- variable to hold the new values
  l_newcolset colset;
  l_jsonrow   maxvarchar2;
  l_tempcolvalue maxvarchar2;
begin
```

Data redaction

```
  l_rowset dbms_tf.row_set_t;
  -- variable to hold the number of rows as retrieved
  l_rowcount pls_integer;
  -- variable to hold the number of put columns
  l_putcolcount pls_integer := dbms_tf.get_env().put_columns.count;
  -- variable to hold the new values
  l_newcolset colset;
  l_jsonrow   maxvarchar2;
  l_tempcolvalue maxvarchar2;
begin
  -- fetch rows into a local rowset
  -- at this point the rows will have columns
  -- from the the table/view/query passed in
  dbms_tf.get_row_set(l_rowset, l_rowcount);
  -- for every row in the rowset...
  for rowindx in 1 .. l_rowcount loop
    -- for every column
    --  FUNCTION Row_To_Char(rowset Row_Set_t,
    --                       rid    PLS_INTEGER,
    --                       format PLS_INTEGER default FORMAT_JSON)
```

Data redaction

```
-- fetch rows into a local rowset
-- at this point the rows will have columns
-- from the the table/view/query passed in
dbms_tf.get_row_set(l_rowset, l_rowcount);
-- for every row in the rowset...
for rowindx in 1 .. l_rowcount loop
  -- for every column
  --  FUNCTION Row_To_Char(rowset Row_Set_t,
  --                       rid    PLS_INTEGER,
  --                       format PLS_INTEGER default FORMAT_JSON)
  --           return VARCHAR2;
  l_jsonrow := dbms_tf.row_to_char(l_rowset, rowindx);
  for colindx in 1 .. l_putcolcount loop
    -- get the value from the column
    l_tempcolvalue := json_value(l_jsonrow, '$.' ||
                        trim('"' from dbms_tf.get_env().get_columns(colindx).name));
    -- redact the value by replacing the first x characters by an *
    l_newcolset(colindx)(rowindx) := lpad('*', redact_first, '*') ||
                                     substr(l_tempcolvalue, redact_first + 1);
  end loop; -- every column
```

Data redaction

```
    --          return VARCHAR2;
    l_jsonrow := dbms_tf.row_to_char(l_rowset, rowindx);
    for colindx in 1 .. l_putcolcount loop
      -- get the value from the column
      l_tempcolvalue := json_value(l_jsonrow, '$.' ||
                          trim('"' from dbms_tf.get_env().get_columns(colindx).name));
      -- redact the value by replacing the first x characters by an *
      l_newcolset(colindx)(rowindx) := lpad('*', redact_first, '*') ||
                                       substr(l_tempcolvalue, redact_first + 1);
    end loop; -- every column
  end loop; -- every row in the rowset
  -- add the newly populated columns to the rowset
  for indx in 1 .. l_putcolcount loop
    dbms_tf.put_col(columnid => indx, collection => l_newcolset(indx));
  end loop;
end;
end redaction_ptf;
/
sho err
```

Data redaction

```
-- create a 'wrapper' function for the polymorphic table function
create or replace function redaction_fnc(p_tbl       in table
                                        ,cols        columns default null
                                        ,redact_first in number default 0) return table
  pipelined row polymorphic using redaction_ptf;
/
```

Data redaction

```
select empno, ename, contract
  from redaction_fnc(emp_contract, columns(contract),14)
/
```

```
     EMPNO ENAME      CONTRACT
---------- ---------- --------------------
      7369 SMITH      **************1050
      7499 ALLEN      **************3840
      7521 WARD       **************3060
      7566 JONES      **************2250
      7654 MARTIN     **************9280
      7698 BLAKE      **************4570
      7782 CLARK      **************7279
      7788 SCOTT      **************6280
      7839 KING       **************3690
      7844 TURNER     **************3790
      7876 ADAMS      **************9620
      7900 JAMES      **************1360
      7902 FORD       **************4750
      7934 MILLER     **************1430

14 rows selected
```

More…

- Polymorphic Table Functions

[http://blog.bar-solutions.com/?p=820]

- Polymorphic Table Functions Part 2

[http://blog.bar-solutions.com/?p=832]

- Oracle Live SQL - Script:
  - Polymorphic Table Function Split Column

[http://bit.ly/PTF_Split]

  - Polymorphic Table Function Split Column Variable Column Name

[http://bit.ly/PTF_Split2]

  - Polymorphic Table Function Split Column Variable Column Name Variable Separator

[http://bit.ly/PTF_Split3]


Q&A

Oracle Cloud Infrastructure

New Free Tier

oracle.com/cloud/free

Always Free

Services you can use for unlimited time

+

30-Day Free Trial

Free credits you can use for more services