Polymorphic Table Functions in 18c

Andrej Pashchenko

Blog

Andrej_SQL

blog.sqlora.com

About me

- Senior Consultant at Trivadis GmbH, Düsseldorf
- Focus Oracle
  - Data Warehousing
  - Application Development
  - Application Performance
- Course instructor „Oracle New Features for Developer“
- Blog: https://blog.sqlora.com

Follow @Andrej_SQL

Table functions in Oracle before 18c

- Table functions are known since 8i

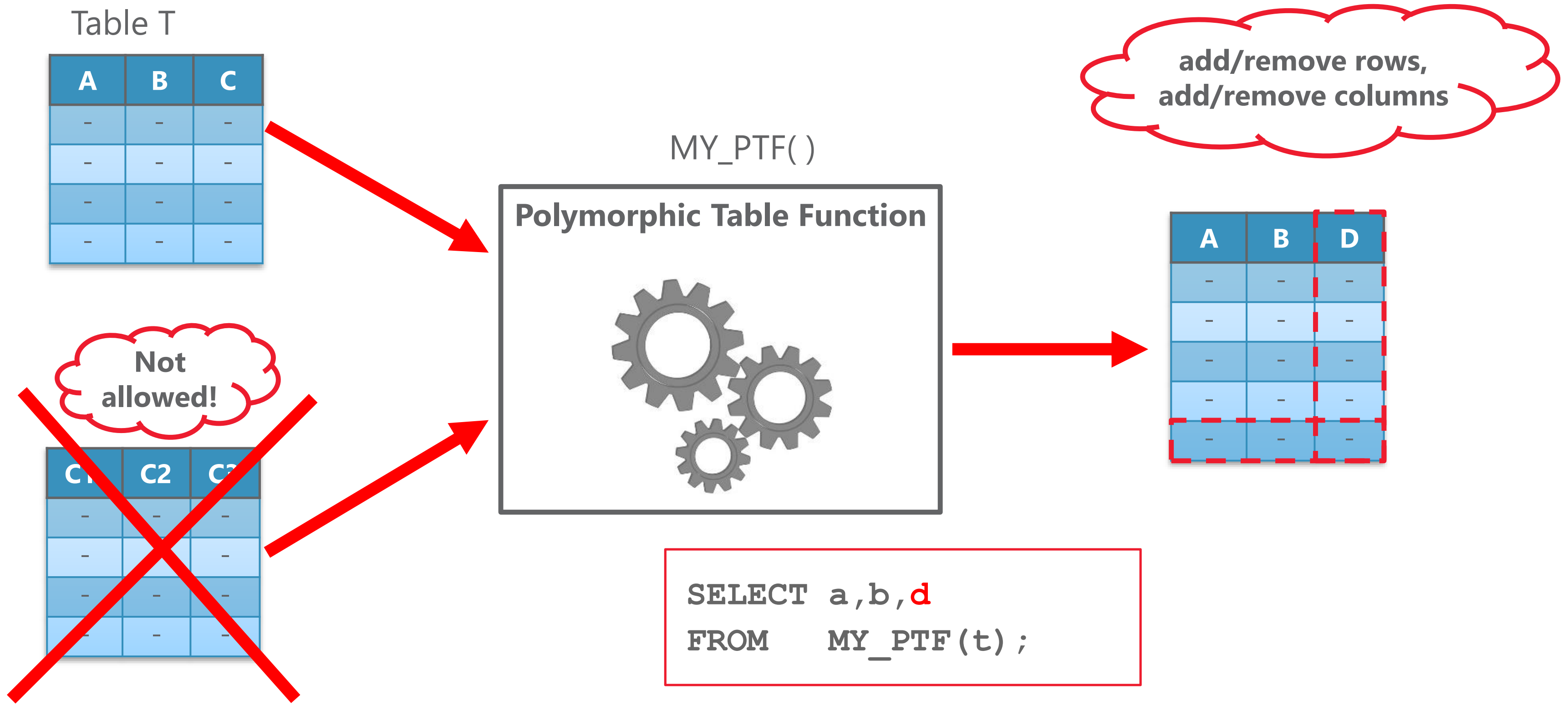
```
CREATE OR REPLACE TYPE names_t IS TABLE OF VARCHAR2 (100);

SELECT t.*
FROM   TABLE(get_emp_names()) t;

SMITH
ALLEN
WARD
JONES
```

- The return collection type **must be** declared beforehand
- Forwarding the data stream into the function in a SQL query is not trivial (CURSOR parameter).
- User defined aggregate functions using Oracle Data Cartridge Interface (ODCI)

Polymorphic Table Functions (PTF)

- Evolution of table functions: part of the ANSI SQL2016 standard
- Abstract complicated business logic and make it available generically at SQL level
- SELECT from the function in the FROM clause
- Like a view, but more procedural and dynamic (without dynamic SQL)
- Polymorphic: a PTF supports different input and output data structures
  - can accept generic table parameters whose structure does not have to be known on definition. Oracle: exactly one table parameter is mandatory in 18c
  - the return table type does not have to be declared on definition, it depends on the input structure and additional function parameters.
- Oracle provides the infrastructure in the **DBMS_TF** package and a new SQL pseudo-operator **COLUMNS().**

Polymorphic Table Functions (PTF)




```
SELECT a,b,d
FROM   MY_PTF(t);
```

Example Task for the First PTF

- Keep only a defined list of columns of the source table in the output.
- Concatenate the remaining (hidden) columns with a separator and display them as a new column.

```
SELECT a,b,d FROM   MY_PTF(t, COLUMNS(a,b));
```

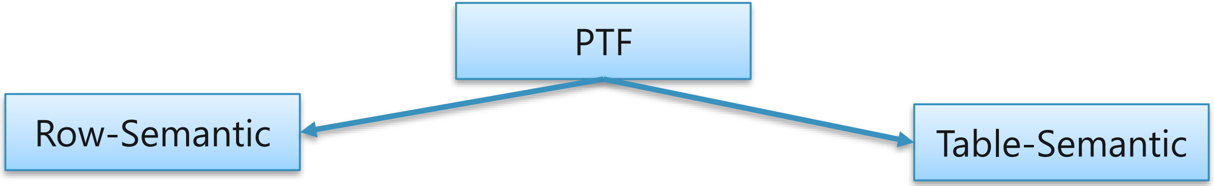
A	B	C	V
1	2	3	ONE
4	5	6	TWO
7	8	9	THREE

→ **MY_PTF** →

A	B	D
1	2	3;ONE
4	5	6;TWO
7	8	9;THREE

trivadis

Types of PTF



Row-Semantic

- Each result row can be derived exclusively from the current source row.
- Use cases:
  - Adding new calculated columns
  - Reformat the data set
  - Output in a special format (JSON, CSV, etc.)
  - Replication
  - Pivot / Transpose

Fits to our task

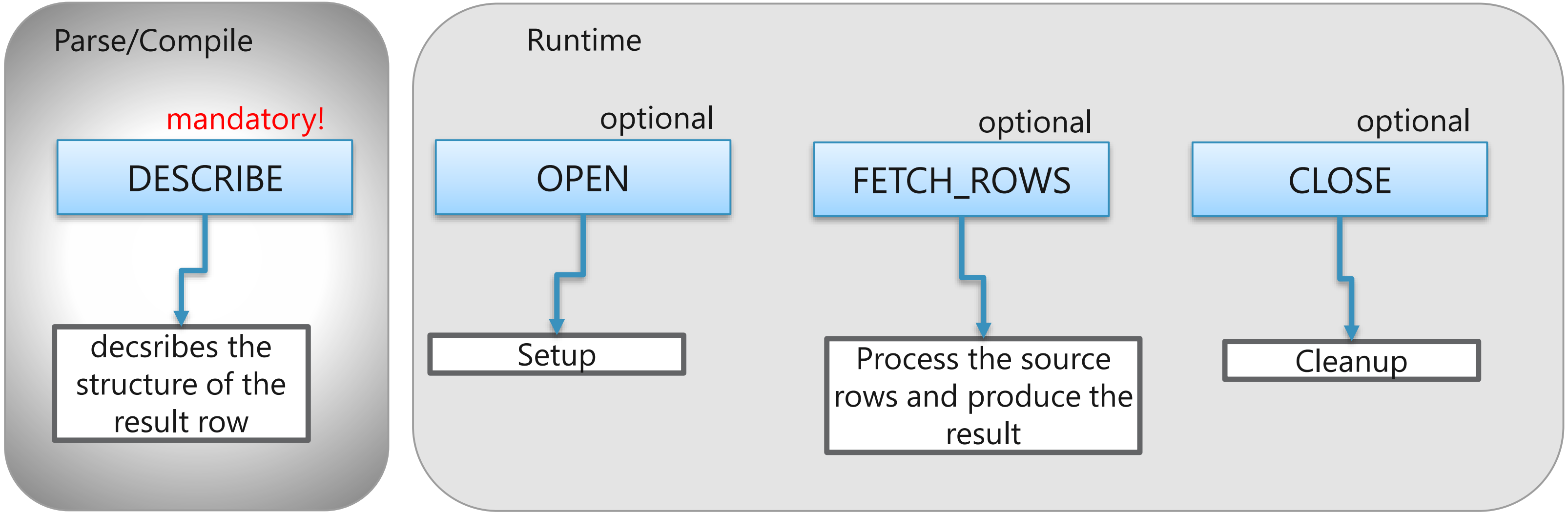
Table-Semantic

- The output row results by looking at the current source row and already processed rows
- Works on the whole table or a (logical) partition of it
- Input table can optionally be partitioned and sorted
- Use cases:
  - User-defined aggregate or window functions

trivadis

Interface Methods

- Each PTF needs an implementation package that provides the implementation of the interface methods:



Definition: Package and PTF

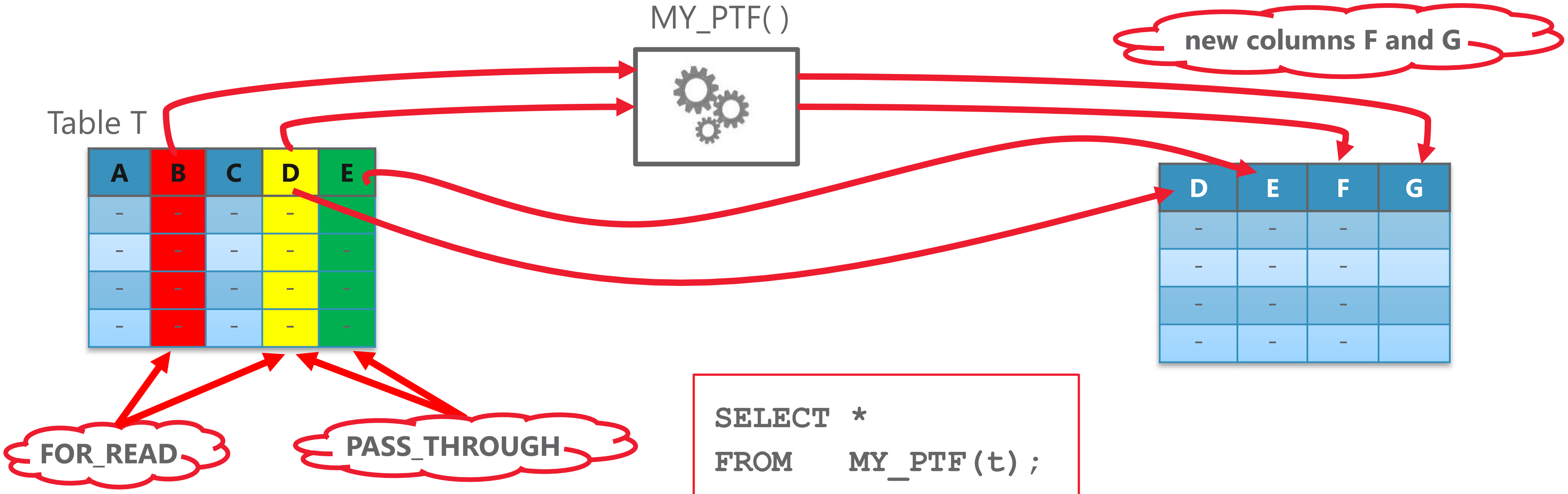
```
CREATE OR REPLACE PACKAGE my_ptf_package AS

FUNCTION describe (tab IN OUT dbms_tf.table_t, cols2stay IN dbms_tf.columns_t )
         RETURN dbms_tf.describe_t;

PROCEDURE fetch_rows;

-- Not required for now
--PROCEDURE open;
--PROCEDURE close;

END my_ptf_package;
/

CREATE OR REPLACE  FUNCTION my_ptf (tab TABLE, cols2stay COLUMNS )
    RETURN TABLE PIPELINED ROW POLYMORPHIC USING my_ptf_package;
```

column list: these columns should remain

DESCRIBE

- The method is called by the database when parsing a SQL and it cannot be invoked explicitly.
- Defines the structure of the result set based on:
  - the structure of the input table
  - other passed parameters
- The database converts the table metadata to DBMS_TF.TABLE_T and any existing columns parameters to DBMS_TF.COLUMNS_T.
- Returns the metadata about the new columns - DBMS_TF.DESCRIBE_T
- The columns of the input table can be marked as **"pass-through"** or **"for read"** (DBMS_TF.TABLE_T is an IN OUT parameter)

PASS-THROUGH and FOR READ Columns




```
SELECT *
FROM   MY_PTF(t);
```

PASS-THROUGH and FOR READ Columns

PASS THROUGH

- passed unchanged to the result set
- defaults:
  - row semantic – all columns
  - table semantic – no columns, except partitioning clause*

FOR READ

- only these columns can be retrieved in FETCH_ROWS.
- default - no columns

- Both properties are NOT mutually exclusive!

The columns in the parameter COLS2STAY are PASS_THROUGH, all others are FOR_READ

* - not working in 18c. Bug?

Implementing DESCRIBE

```
…
FUNCTION describe (tab IN OUT dbms_tf.table_t, cols2stay IN dbms_tf.columns_t )
         RETURN dbms_tf.describe_t IS
  new_col_name CONSTANT VARCHAR2(30) := 'AGG_COL';
BEGIN
  FOR I IN 1 .. tab.COLUMN.COUNT LOOP
     IF NOT tab.COLUMN(i).description.name MEMBER OF cols2stay THEN
          tab.column(i).pass_through := false;
          tab.column(i).for_read := true;
     END IF;
  END LOOP;

  RETURN dbms_tf.describe_t( new_columns =>
               dbms_tf.columns_new_t( 1 => dbms_tf.column_metadata_t(
                           name => new_col_name,
                           TYPE => dbms_tf.type_varchar2)));
END;
…
```

Hide and aggregate these columns

Default=true for all others

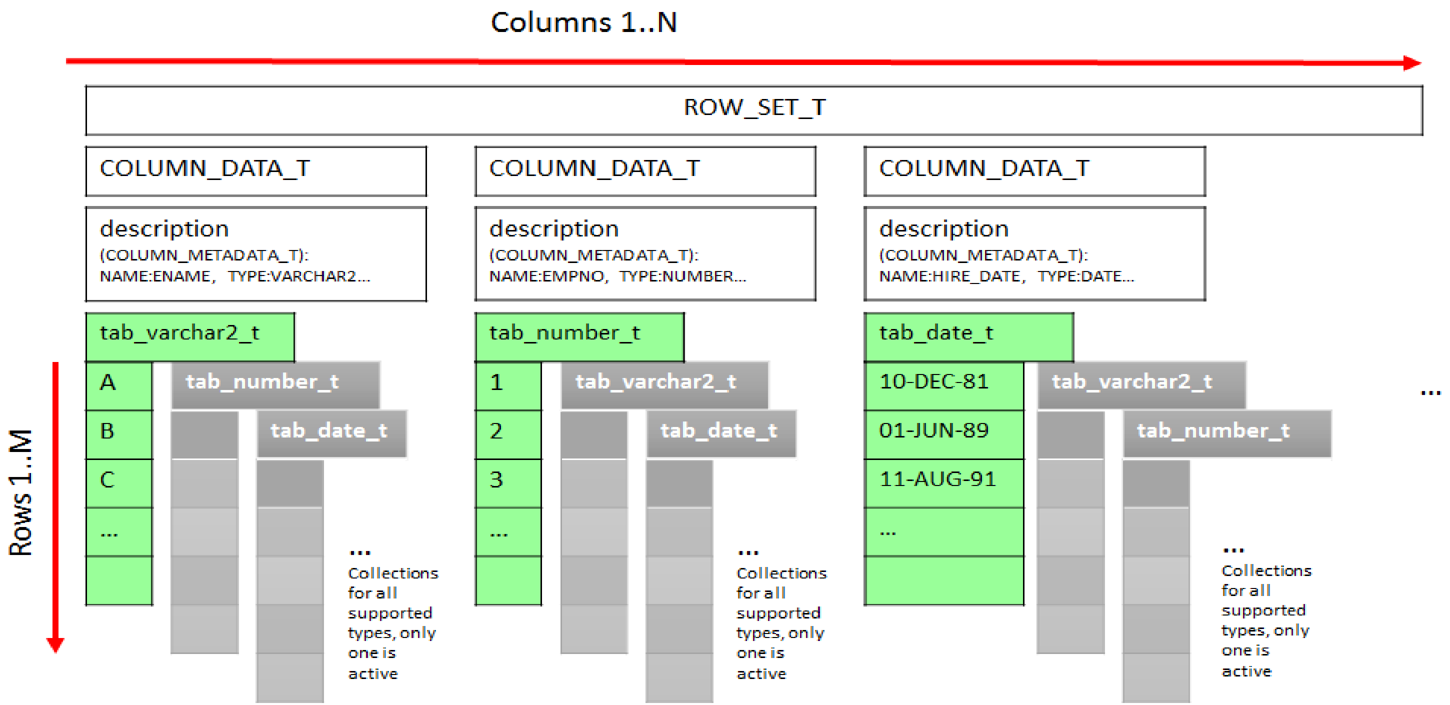
Metadata about the new columns

trivadis

Implementing FETCH_ROWS

Context DESCRIBE		Context FETCH_ROWS
FOR READ Columns	=	GET-Columns
New Columns	=	PUT-Columns
PASS_THROUGH columns	≠	Not visible in FETCH_ROWS

The Structure of ROWSET



Implementation of FETCH_ROWS

```
…
PROCEDURE fetch_rows IS
  rowset dbms_tf.row_set_t;
  colcnt PLS_INTEGER;
  rowcnt PLS_INTEGER;
  agg_col  dbms_tf.tab_varchar2_t;
BEGIN
    dbms_tf.get_row_set(rowset, rowcnt, colcnt);
    FOR r IN 1..rowcnt LOOP
      agg_col(r) := '';
      FOR c IN 1..colcnt LOOP
       agg_col(r) := agg_col(r)||';'||DBMS_TF.COL_TO_CHAR(rowset(c), r);
      END LOOP;
      agg_col(r) := ltrim (agg_col(r) ,';');
    END LOOP;
    dbms_tf.put_col(1, agg_col);
END;
…
```

fetch the read columns

„aggregate“

return the Data for the new column

trivadis

Polymorphism in Action

```
SQL> SELECT * FROM my_ptf(t, COLUMNS(A));

         A AGG_COL
---------- --------------------------------------------------
         1 2;3;"ONE"
         4 5;6;"TWO"
         7 8;9;"THREE"

SQL> SELECT * FROM my_ptf(t, COLUMNS(A,B));

         A          B AGG_COL
---------- ---------- --------------------------------------------------
         1          2 3;"ONE"
         4          5 6;"TWO"
         7          8 9;"THREE"

SQL> SELECT * FROM my_ptf(scott.emp, COLUMNS(empno));

     EMPNO AGG_COL
---------- --------------------------------------------------
      7369 "SMITH";"CLERK";7902;"17-DEC-80";800;;20
      7499 "ALLEN";"SALESMAN";7698;"20-FEB-81";1600;300;30
      7521 "WARD";"SALESMAN";7698;"22-FEB-81";1250;500;30
```

Task 2 = Task 1 + JSON-Format

- Keep only a defined list of columns of the source table in the output.
- Return the remaining columns as JSON document

```
SELECT a,b, json_col FROM   split_json(t, COLUMNS(a,b));
```

A	B	C	V
1	2	3	ONE
4	5	6	TWO
7	8	9	THREE

→ **SPLIT_JSON** →

A	B	JSON_COL
1	2	{"C":3, "V":"ONE"}
4	5	{"C":6, "V":"TWO"}
7	8	{"C":9, "V":"THREE"}

Implementation for Task 2

```
…
FUNCTION describe …

PROCEDURE fetch_rows IS
  rowset dbms_tf.row_set_t;
  rowcnt PLS_INTEGER;
  json_col  dbms_tf.tab_varchar2_t;
BEGIN
    dbms_tf.get_row_set(rowset, rowcnt);
    FOR r IN 1..rowcnt LOOP
      json_col(r) := DBMS_TF.ROW_TO_CHAR(rowset, r, DBMS_TF.FORMAT_JSON);
    END LOOP;
    dbms_tf.put_col(1, json_col);
END;…
```

DESCRIBE is still the same…

Return the row set as JSON

SPLIT_JSON

```
SQL> SELECT * FROM split_json(t, COLUMNS(A));

         A JSON_COL
---------- ----------------------------------------
         1 {"B":2, "C":3, "V":"ONE"}
         4 {"B":5, "C":6, "V":"TWO"}
         7 {"B":8, "C":9, "V":"THREE"}

SQL> SELECT * FROM split_json(t, COLUMNS(A,B));

         A          B JSON_COL
---------- ---------- --------------------------
         1          2 {"C":3, "V":"ONE"}
         4          5 {"C":6, "V":"TWO"}
         7          8 {"C":9, "V":"THREE"}

SQL> SELECT * FROM split_json(scott.emp, COLUMNS(empno));

     EMPNO JSON_COL
---------- ----------------------------------------------------------------
      7369 {"ENAME":"SMITH", "JOB":"CLERK", "MGR":7902, "HIREDATE":"17-DEC-80", ...
      7499 {"ENAME":"ALLEN", "JOB":"SALESMAN", "MGR":7698, "HIREDATE":"20-FEB-81", ...
      7521 {"ENAME":"WARD", "JOB":"SALESMAN", "MGR":7698, "HIREDATE":"22-FEB-81", ...
```

Task 3: Transpose the Columns to Rows

- Keep only a defined list of columns of the source table in the output to identify the rows.
- Return the remaining columns as key-value pairs

More rows in the result than in the source?

```
SELECT * FROM   tab2keyval(t, COLUMNS(a,b));
```

A	B	C	V
1	2	3	ONE
4	5	6	TWO
7	8	9	THREE

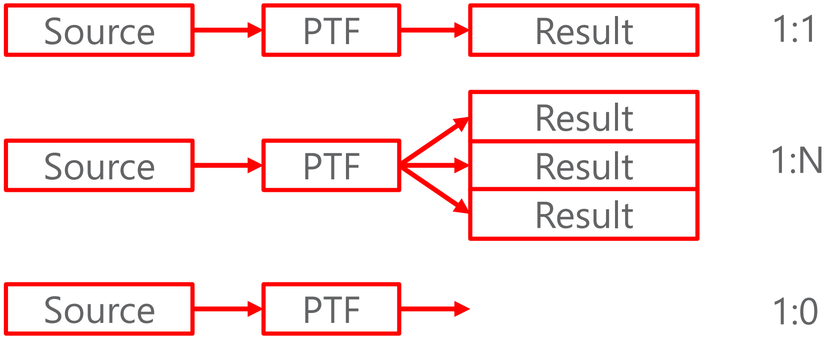
→ **MY_PTF** →

A	B	KEY_NAME	KEY_VALUE
1	2	C	3
1	2	V	ONE
4	5	C	6
4	5	V	TWO
7	8	C	9
7	8	V	THREE

trivadis

Row Replication

- Row replication allows to multiply or hide records
- DBMS_TF.ROW_REPLICATION
- As a fixed factor for all rows
- Or a value per row
- A flag have to be set in DESCRIBE, ORA-62574 otherweise
- At the moment it is the only way to add new records. But pay attention to PASS_THROUGH columns!



Implementation for Task 3

```
…
FUNCTION describe (tab IN OUT dbms_tf.table_t, cols2stay IN dbms_tf.columns_t )
         RETURN dbms_tf.describe_t IS
BEGIN
  FOR I IN 1 .. tab.COLUMN.COUNT LOOP
     IF NOT tab.COLUMN(i).description.name MEMBER OF cols2stay THEN
          tab.column(i).pass_through := false;
          tab.column(i).for_read := true;
     END IF;
  END LOOP;

  RETURN dbms_tf.describe_t(new_columns =>
               dbms_tf.columns_new_t( 1 => dbms_tf.column_metadata_t(
                           name => 'KEY_NAME', TYPE => dbms_tf.type_varchar2),
                           2 => dbms_tf.column_metadata_t(
                           name => 'KEY_VALUE', TYPE => dbms_tf.type_varchar2)),
            row_replication => true);
END;
…
```

Set the row-replication flag, otherwise ORA-62574

Implementation for Task 3 - FETCH_ROWS

```
…
PROCEDURE fetch_rows IS
  rowset dbms_tf.row_set_t;
  repfac dbms_tf.tab_naturaln_t;
  rowcnt PLS_INTEGER;
  colcnt PLS_INTEGER;
  name_col  dbms_tf.tab_varchar2_t;
  val_col  dbms_tf.tab_varchar2_t;
  env    dbms_tf.env_t := dbms_tf.get_env();
BEGIN
    dbms_tf.get_row_set(rowset, rowcnt, colcnt);
    FOR i IN 1 .. rowcnt LOOP
      repfac(i) := 0;
    END LOOP;
...
```

Collections for new columns

Environment information

trivadis

Implementation for Task 3 - FETCH_ROWS

```
...
    FOR r IN 1..rowcnt LOOP
      FOR c IN 1..colcnt LOOP
        repfac(r) := repfac(r) + 1;
        name_col(nvl(name_col.last+1,1)) :=
             INITCAP( regexp_replace(env.get_columns(c).name, '^"|"$'));
        val_col(nvl(val_col.last+1,1)) := DBMS_TF.COL_TO_CHAR(rowset(c), r);
      END LOOP;
    END LOOP;
    dbms_tf.row_replication(replication_factor => repfac);
    dbms_tf.put_col(1, name_col);
    dbms_tf.put_col(2, val_col);
END;
…
```

Duplicate the row for each column to be transposed.

Set the replication factor

TAB2KEYVAL

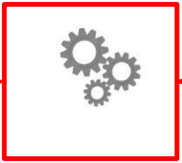
```
SQL> SELECT * FROM tab2keyval(t, COLUMNS(A,B));

         A          B KEY_NAME   KEY_VALUE
---------- ---------- ---------- --------------------
         1          2 C          3
         1          2 V          "ONE"
         4          5 C          6
         4          5 V          "TWO"
         7          8 C          9
         7          8 V          "THREE"


SQL> SELECT * FROM tab2keyval(scott.emp, COLUMNS(empno));

     EMPNO KEY_NAME   KEY_VALUE
---------- ---------- --------------------
      7369 Ename      "SMITH"
      7369 Job        "CLERK"
      7369 Mgr        7902
      7369 Hiredate   "17-DEC-80"
      7369 Sal        800
...
```


trivadis

Task 4: The Length of the CSV representation

- Based on Task 1, calculate the lengths of the CSV representation for the whole table or logical partition

DEPTNO	AGG_COL
10	7934;"MILLER";"CLERK";778...
20	7902;"FORD";"ANALYST";756...
30	7900;"JAMES";"CLERK";7698...
20	7876;"ADAMS";"CLERK";7788...



DEPTNO	AGG_COL	ROW_LEN	SUM_LEN
10	7934;"MILLER";"CLERK"...	44	44
20	7902;"FORD";"ANALYST"...	44	88
30	7900;"JAMES";"CLERK";...	42	130
20	7876;"ADAMS";"CLERK";...	43	173

Table Semantic PTF!

trivadis

Execution of FETCH_ROWS

- FETCH_ROWS is executed 0 to N times
- Data is processed in rowsets
- The size of the rowset is calculated at runtime (<= 1024)
- Can also be executed in parallel
  - Row Semantics can be parallelized by the DB without restrictions
  - Table Semantics: parallelization based on the PARTITION BY clause
- The developer always works with the "active" rowset only
- The required intermediate results can be stored in execution store (XSTORE) or package structures protected by execution ID (XID).

trivadis

Implementation for Task 4

```
…
FUNCTION describe (tab IN OUT dbms_tf.table_t, cols2sum IN dbms_tf.columns_t )
         RETURN dbms_tf.describe_t IS
BEGIN
  FOR I IN 1 .. tab.COLUMN.COUNT LOOP
     IF tab.COLUMN(i).description.name MEMBER OF cols2sum THEN
         tab.column(i).for_read := true;
     END IF;
  END LOOP;

  RETURN dbms_tf.describe_t(
           new_columns => dbms_tf.columns_new_t(
                  1 => dbms_tf.column_metadata_t(
                           name => 'LEN',  TYPE => dbms_tf.type_number),
                  2 => dbms_tf.column_metadata_t(
                           name => 'SUM_LEN', TYPE => dbms_tf.type_number)));
END;
…
```

Implementation for Task 4

```
PROCEDURE fetch_rows IS
  ...
  len_col  dbms_tf.tab_number_t;
  len_curr_col  dbms_tf.tab_number_t;
  v_len pls_integer;
  v_currlen pls_integer := 0;
BEGIN
    dbms_tf.get_row_set(rowset, rowcnt, colcnt);
    dbms_tf.xstore_get('len', v_len);
    v_currlen := nvl(v_len, 0);
    FOR r IN 1..rowcnt LOOP
      len_col(r) := length (rowset(1).tab_varchar2(r));
      v_currlen := v_currlen + len_col(r);
      len_curr_col(r) := v_currlen ;
    END LOOP;
    dbms_tf.xstore_set('len', v_len+v_currlen);
    dbms_tf.put_col(1, len_col);
    dbms_tf.put_col(2, len_curr_col);
END;
```

Reading the intermediate result from the Execution Store

Saving the intermediate result in the Execution Store

…

trivadis

SUMLEN_PTF

```
SQL> select * from sumlen_ptf(my_ptf(scott.emp, COLUMNS(deptno)), columns(agg_col));

ORA-62569: nested polymorphic table function is disallowed

SQL> with agg as (SELECT * FROM my_ptf(scott.emp, COLUMNS(deptno)))
  2  select * from sumlen_ptf(agg partition by deptno, columns(agg_col));

    DEPTNO AGG_COL                                               LEN    SUM_LEN
---------- --------------------------------------------------- ---------- ----------
        10 7782;"CLARK";"MANAGER";7839;"09-JUN-81";2450;               45         45
        10 7839;"KING";"PRESIDENT";;"17-NOV-81";5000;                  42         87
        10 7934;"MILLER";"CLERK";7782;"23-JAN-82";1300;                44        131
        20 7566;"JONES";"MANAGER";7839;"02-APR-81";2975;               45         45
        20 7902;"FORD";"ANALYST";7566;"03-DEC-81";3000;                44         89
        20 7876;"ADAMS";"CLERK";7788;"23-MAY-87";1100;                 43        132
        20 7369;"SMITH";"CLERK";7902;"17-DEC-80";800;                  42        174
        20 7788;"SCOTT";"ANALYST";7566;"19-APR-87";3000;               45        219
        30 7521;"WARD";"SALESMAN";7698;"22-FEB-81";1250;500            48         48
        30 7844;"TURNER";"SALESMAN";7698;"08-SEP-81";1500;0            48         96
        30 7499;"ALLEN";"SALESMAN";7698;"20-FEB-81";1600;300           49        145
        30 7900;"JAMES";"CLERK";7698;"03-DEC-81";950;                  42        187
        30 7698;"BLAKE";"MANAGER";7839;"01-MAY-81";2850;               45        232
        30 7654;"MARTIN";"SALESMAN";7698;"28-SEP-81";1250;140          51        283
...
```

the input is partitioned

trivadis

Predicate Pushdown

- If semantically possible, predicates, projections, partitions are passed to the underlying table.
- Only possible with PASS_THROUGH and PARTITION BY columns

```
SELECT * FROM my_ptf(scott.emp, COLUMNS(deptno)) WHERE deptno = 10

---------------------------------------------------------------------------------------------
| Id  | Operation                              | Name               | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |                    |       |       |     2 (100)|          |
|   1 |  POLYMORPHIC TABLE FUNCTION            | MY_PTF             |     3 |   261 |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID BATCHED| EMP                |     3 |   114 |     2   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN                    | SCOTT_DEPTNO_IDX |     3 |       |     1   (0)| 00:00:01 |
---------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("EMP"."DEPTNO"=10)
```


trivadis

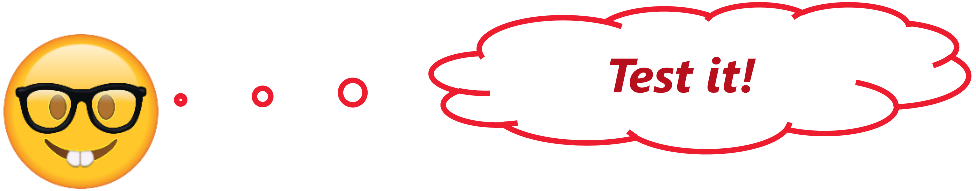
TOPNPLUS

- Return the Top-N records and an aggregation of the remaining rows in a single query.
- Not yet 100% properly implementable in 18.3

```
SQL> SELECT   deptno, empno, ename, job, sal
  2  FROM     topnplus(scott.emp partition by deptno order by sal desc, COLUMNS(sal), columns(deptno), 2)

    DEPTNO      EMPNO ENAME      JOB              SAL
---------- ---------- ---------- ---------- ----------
        10       7839 KING       PRESIDENT        5000
        10       7782 CLARK      MANAGER          2450
        10                                        1300
        20       7788 SCOTT      ANALYST          3000
        20       7902 FORD       ANALYST          3000
        20                                        4875
        30       7698 BLAKE      MANAGER          2850
        30       7499 ALLEN      SALESMAN         1600
        30                                        4950
```

Conclusion

- Powerful and flexible extension for SQL
- Clearly defined interface to the DB lets the developer do his job: more business logic, less technical details
- Performance optimizations right from the beginning
- ANSI SQL but not all PTF features from ANSI SQL 2016 implemented yet
- No real support for adding new rows yet
- Real aggregate functions not yet 100% possible
- Partly contradictory documentation



trivadis

http://blog.sqlora.com/en/tag/ptf/

http://standards.iso.org/ittf/PubliclyAvailableStandards/c069776_ISO_IEC_TR_19075-7_2017.zip - document on PTF (ISO/IEC TR 19075-7:2017)