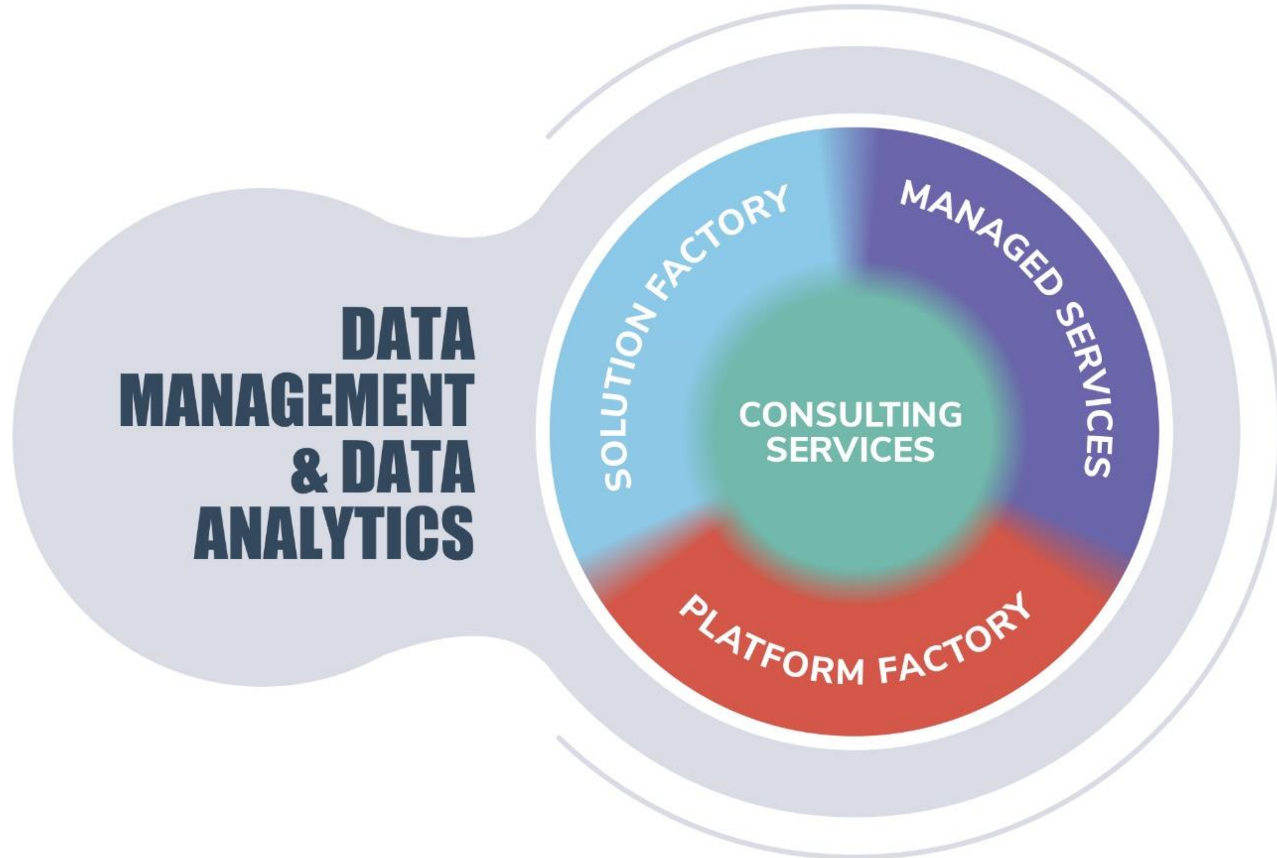
Polymorphic Table Functions in 18c

Andrej Pashchenko

@Andrej_SQL doag2018

trivadis
makes IT easier.

We help to generate added value from data



ivadis
makes IT easier.

With over 650 specialists and IT experts in your region.

- 16 Trivadis branches and more than 650 employees
- Experience from more than 1,900 projects per year at over 800 customers
- 250 Service Level Agreements
- Over 4,000 training participants
- Research and development budget: CHF 5.0 million
- Financially self-supporting and sustainably profitable

trivadis
makes IT easier.

Über mich

- Senior Consultant bei der Trivadis GmbH, Düsseldorf
- Schwerpunkt Oracle
  - Data Warehousing
  - Application Development
  - Application Performance
- Kurs-Referent „Oracle 12c New Features für Entwickler“ und „Beyond SQL and PL/SQL“

- Blog: http://blog.sqlora.com

trivadis makes IT easier.

Tabellenfunktionen vor 18c

- Tabellenfunktionen schon länger bekannt (8i)

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

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

SMITH
ALLEN
WARD
JONES
```

- Der Collection Typ der Rückgabe **muss vorher** deklariert werden
- Das Weiterleiten des Datenstroms in die Funktion in einer SQL-Abfrage ist nicht trivial (CURSOR-Parameter)
- Eigene Aggregatfunktionen mit Oracle Data Cartridge Interface (ODCI)

trivadis
makes IT easier.

Polymorphe Tabellenfunktionen (PTF)

- Evolution von Tabellenfunktionen: Teil von ANSI SQL2016
- Komplizierte Geschäftslogik abstrahieren und generisch auf SQL-Ebene zur Verfügung zu stellen
- SELECT aus der Funktion in der FROM-Klausel
- Wie eine View, nur prozeduraler und dynamischer (ohne dynamisches SQL)
- Polymorph: eine PTF unterstützt unterschiedliche Eingabe- und Ausgabe-Datenstrukturen:
  - Kann generische Tabellenparameter akzeptieren, deren Datensatzstruktur bei der Definition nicht bekannt sein muss. Oracle: genau ein Tabellenparameter ist Pflicht in 18c
  - Der Rückgabe-Tabellentyp muss nicht bei der Definition deklariert sein, hängt von der Eingabestruktur und zusätzlichen Funktionsparametren ab

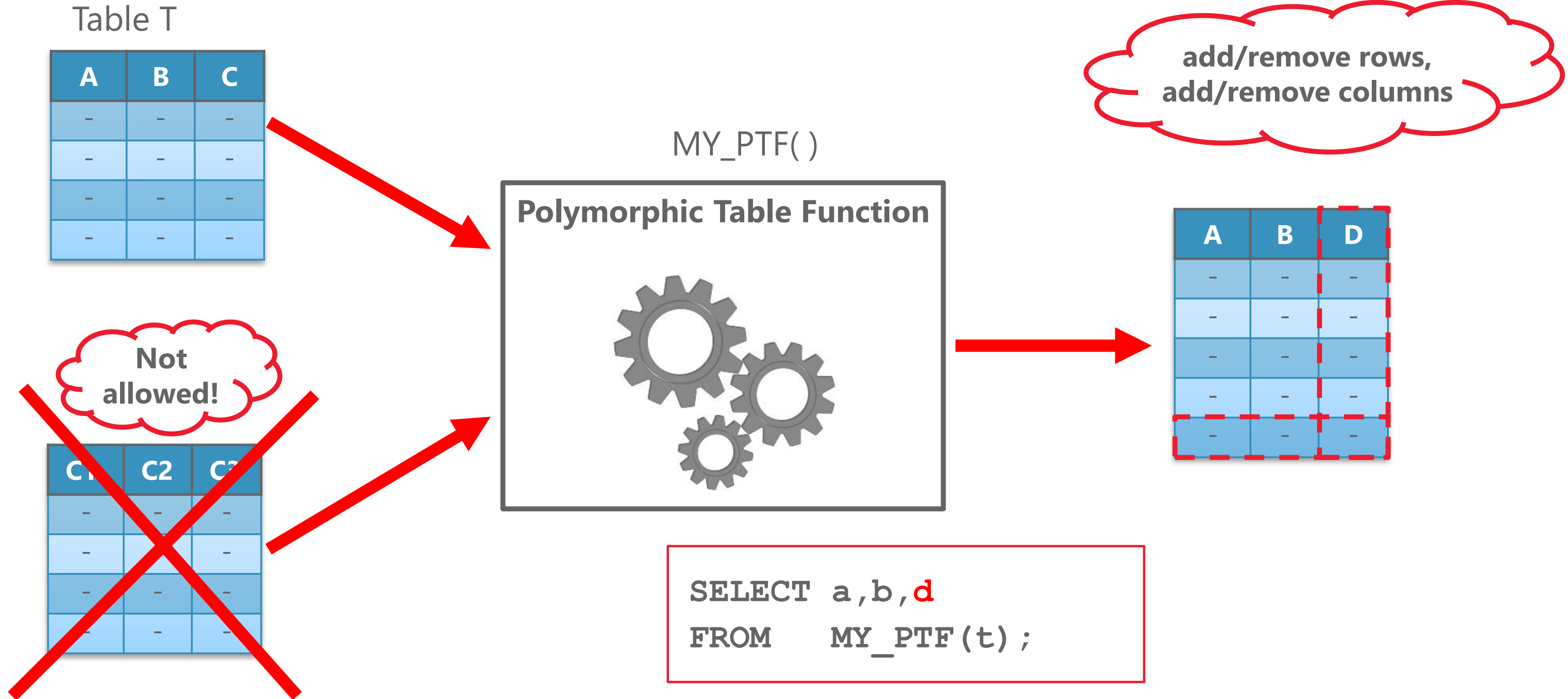
trivadis makes IT easier.

Polymorphe Tabellenfunktionen (PTF)

- Rollen:
  - **PTF-Entwickler** implementiert den prozeduralen Mechanismus für die Lösung der Anforderungen an die PTF und veröffentlicht die Schnittstelle
  - **Abfrage-Entwickler** ruft die PTF in der SQL-Abfrage auf
  - **DBMS** ist für Kompilierung (Parsing), Ausführung von SQL zuständig und übernimmt viele technische Aspekte: parallele Ausführung, Datenübergabe in die / aus der Funktion, Zustandsverwaltung, etc.
- Oracle stellt dem PTF-Entwickler die Infrastruktur im Package **DBMS_TF** sowie einen neuen SQL-Operator **COLUMNS()** zur Verfügung
- PTF-Entwickler kann sich auf der fachlichen Aufgabe konzentrieren.

trivadis makes IT easier.

Polymorphe Tabellenfunktionen (PTF)



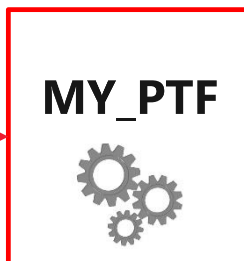
trivadis makes IT easier.

Beispielaufgabe für die erste PTF

- Nur eine definierte Liste der Spalten der Quelltabelle in der Ausgabe behalten.
- Die restlichen (versteckten) Spalten mit einem Trennzeichen zusammenfügen und als neue Spalte darstellen

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

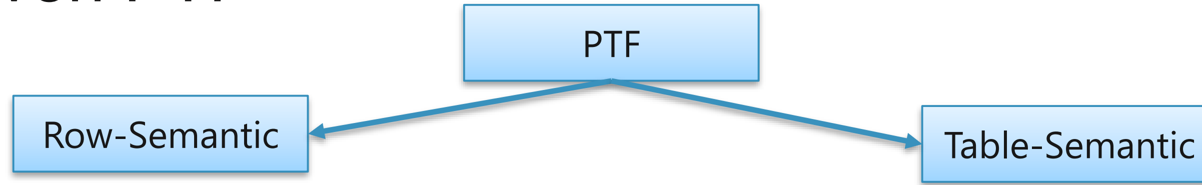
A	B	C	V
1	2	3	ONE
4	5	6	TWO
7	8	9	THREE



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

trivadis makes IT easier.

Typen von PTF



- Jeder Ergebnis-Datensatz kann ausschließlich aus dem laufenden Quell-Datensatz abgeleitet werden
- Anwendungsbeispiele:
  - Hinzufügen neuer berechneter Spalten
  - Umformatieren des Datensatzes
  - Ausgabe in einem speziellen Format (JSON, CSV, etc.)
  - Replikation
  - Pivot / Transponieren

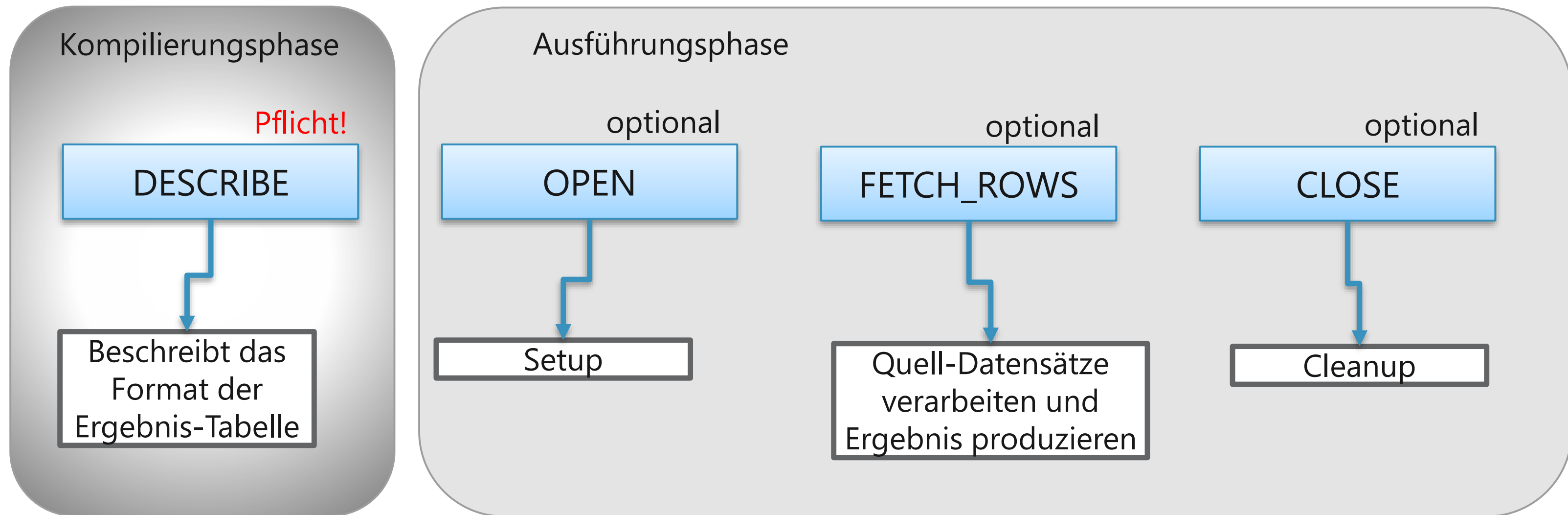
Passt zur Beispielaufgabe

- Das Ergebnis-Datensatz ergibt sich aus der Betrachtung vom laufenden Quell-Datensatz und bereits verarbeiteten Datensätze
- Arbeitet auf der ganzen Tabelle bzw. einer (logischen) Partition davon
- Input-Tabelle kann optional partitioniert und sortiert werden
- Anwendungsbeispiele:
  - Benutzerdefinierte Aggregat- / Fensterfunktionen

trivadis
makes IT easier.

Interface-Methoden

- Jede PTF braucht ein Implementierungs-Package, das die Implementierung der Interface-Methoden bereitstellt:



trivadis
makes IT easier.

Definition: Package und 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;
```

Spaltenliste: diese Spalten wollen wir beibehalten

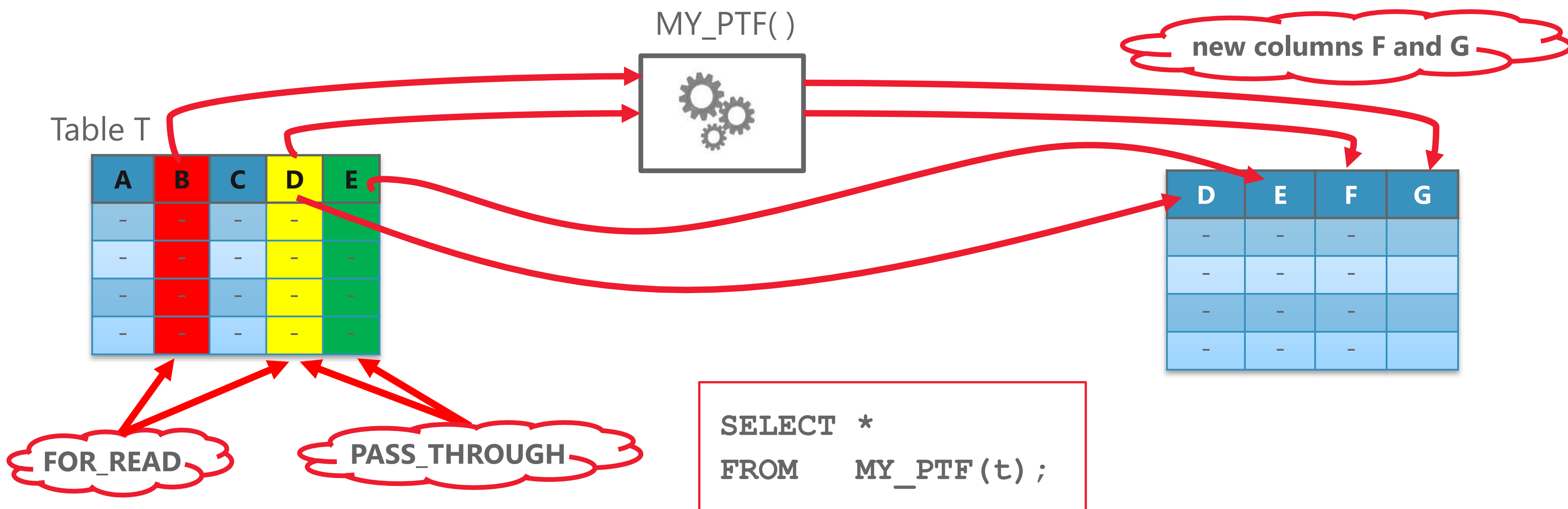
trivadis makes IT easier.

DESCRIBE

- die Methode wird beim Parsen von SQL von der DB aufgerufen, kann nicht explizit aufgerufen werden
- definiert die Struktur der Rückgabe-Tabelle ausgehend von:
  - der Struktur der Input-Tabelle
  - weiteren Parametern
- die DB wandelt die Tabellen-Metadaten in DBMS_TF.TABLE_T, die evtl. vorhandenen Columns-Parameter – in DBMS_TF.COLUMNS_T um
- liefert die Metadaten über die neuen Spalten zurück - DBMS_TF.DESCRIBE_T
- außerdem können die Spalten der Input-Tabelle als **„pass-through“** oder **„for read“** markiert werden (DBMS_TF.TABLE_T ist ein IN OUT Parameter)

trivadis
makes IT easier.

PASS-THROUGH und FOR READ Spalten




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


trivadis makes IT easier.

PASS-THROUGH und FOR READ Spalten

PASS THROUGH

- Werden ohne Änderung in das Ergebnis weitergeleitet
- Defaults:
  - Row Semantic – alle Spalten
  - Table Semantic – keine Spalten, außer Partitioning-Klausel*

FOR READ

- Nur diese Spalten werden in FETCH_ROWS abrufbar.
- Default- keine Spalten

- Beide Eigenschaften schließen sich gegenseitig NICHT aus!

Spalten im Parameter COLS2STAY sind PASS_THROUGH, alle anderen FOR_READ

* - in 18.1 funktioniert es nicht. Bug?

trivadis makes IT easier.

Implementierung 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;
…
```

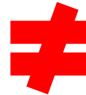
Diese Spalten verstecken und aggregieren

Default=true für alle anderen

Metadaten für die neue Spalte

trivadis makes IT easier.

Implementierung FETCH_ROWS

Kontext DESCRIBE		Kontext FETCH_ROWS
FOR READ Columns		GET-Columns
New Columns		PUT-Columns
PASS_THROUGH columns		Nicht sichtbar in FETCH_ROWS

trivadis
makes IT easier.

Implementierung 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;
…
```

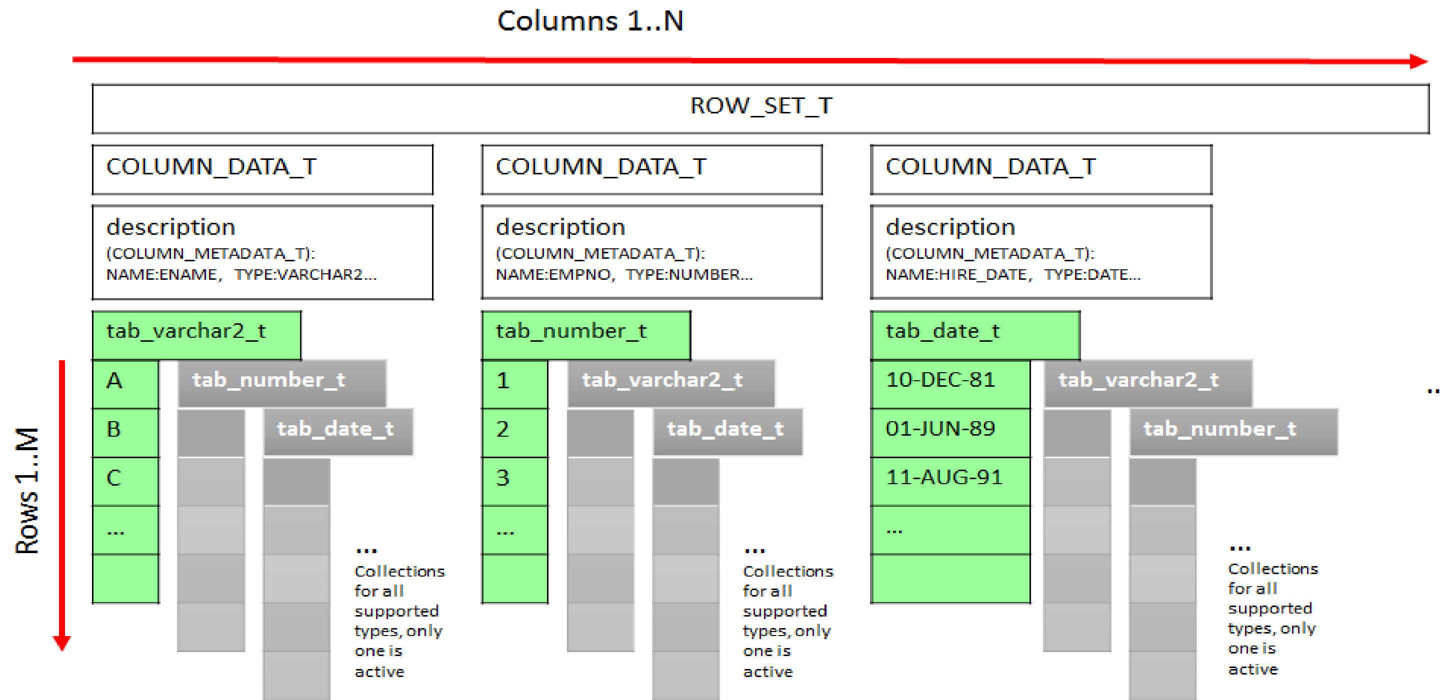
Read Columns fetchen

Aggregation durchführen

Daten für die neue Spalte zurückgeben

trivadis makes IT easier.

Struktur von ROWSET



trivadis makes IT easier.

Polymorphismus in der Praxis

```
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
```

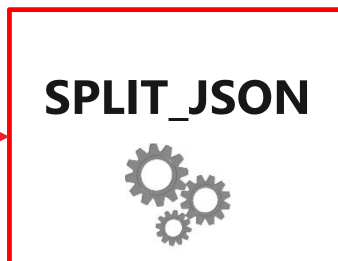

trivadis makes IT easier.

Aufgabe 2 = Aufgabe 1 + JSON-Format

- Nur eine definierte Liste der Spalten der Quelltabelle in der Ausgabe behalten.
- Die restlichen (versteckten) Spalten als JSON-Dokument ausgeben

```
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



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

trivadis
makes IT easier.

Implementierung für Aufgabe 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 ändert sich nicht…

Row Set als JSON ausgeben

trivadis makes IT easier.

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", ...
```


trivadis makes IT easier.

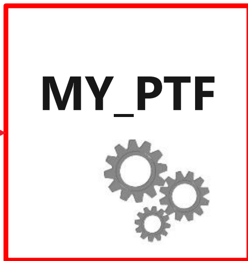
Aufgabe 3: Spalten zu Zeilen transponieren

- Nur eine definierte Liste der Spalten der Quelltabelle zur Identifikation in der Ausgabe behalten.
- Die restlichen Spalten als Key-Value Paare darstellen

Mehr Datensätze im Ergebnis als in der Quelle?

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

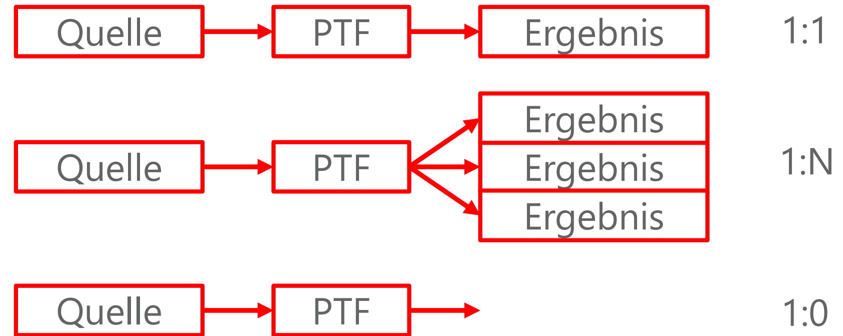
A	B	C	V
1	2	3	ONE
4	5	6	TWO
7	8	9	THREE



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

Row Replication

- Row Replication erlaubt, Datensätze zu multiplizieren oder zu auszublenden
- DBMS_TF.ROW_REPLICATION
- als Faktor für alle Datensätze
- oder auf Datensatz-Basis
- In DESCRIBE muss ein Flag gesetzt werden, sonst ORA-62574
- Im Moment, die einzige Möglichkeit neue Datensätze hinzuzufügen. Aber: PASS_THROUGH Spalten beachten!



trivadis makes IT easier.

Implementierung für Aufgabe 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;
…
```

Row-Repliaction Flag setzen, sonst ORA-62574

trivadis makes IT easier.

Implementierung für Aufgabe 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 für die neuen Spalten

Environment-Informationen

trivadis makes IT easier.

Implementierung für Aufgabe 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;
...
```

Für jede zu transponierende Spalte den Datensatz duplizieren

Replikation setzen

trivadis makes IT easier.

TAB2KEYVAL

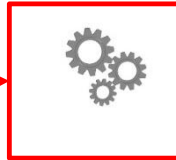
```
SQL> SELECT * FROM tab2keyval(t, COLUMNS(A,B));

         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
makes IT easier.

Aufgabe 4: Gesamtlänge der CSV-Darstellung

- Basierend auf der Aufgabe 1 die Längen der CSV-Darstellung über die gesamte Tabelle bzw. eine Partition berechnen

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

trivadis makes IT easier.

Ausführung FETCH_ROWS

- FETCH_ROWS wird 0 bis N Mal aufgerufen
- Datensätze werden in Rowsets verarbeitet
- Die Größe des Rowsets wird zur Laufzeit errechnet (<= 1024)
- Kann auch parallel ausgeführt werden
  - Row Semantics kann von der DB beliebig parallelisiert werden
  - Table Semantics: Parallelisierung auf der Basis der PARTITION BY-Klausel
- Der Entwickler arbeitet immer mit „aktivem“ Rowset
- Die benötigten Zwischenergebnisse können in Execution Store (XSTORE) abgelegt werden.

trivadis
makes IT easier.

Implementierung für Aufgabe 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),
                  3 => dbms_tf.column_metadata_t(
                           name => 'SUM_LEN', TYPE => dbms_tf.type_number)));
END;
…
```

trivadis
makes IT easier.

Implementierung für Aufgabe 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;
```

…

Das Zwischenergebnis aus dem Execution-Store auslesen

Das Zwischenergebnis im Execution-Store speichern

trivadis makes IT easier.

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
...
```

Input partitioniert

trivadis
makes IT easier.

Predicate Pushdown

- Wenn semantisch möglich, werden Prädikate, Projektionen, Partitionen bis in die unterliegende Tabelle weitergereicht
- Nur möglich mit PASS_THROUGH und PARTITION BY Spalten

```
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 makes IT easier.

TOPNPLUS

- In einer Abfrage die Top-N Datensätze ausgeben, den Rest aggregiert darstellen
- In 18.3 noch nicht 100% sauber umsetzbar

```
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
```

trivadis
makes IT easier.

Fazit

- 😃 Mächtige und flexible Erweiterung für SQL
- 😃 Klar definierte Schnittstelle zur DB lässt den Entwickler seinen Job machen: mehr Geschäftslogik, weniger technische Details
- 😃 Performance Optimierungen von Anfang an
- 😏 ANSI SQL aber noch nicht alle PTF-Features aus ANSI SQL 2016 umgesetzt
- 😏 Noch kein Support für das Hinzufügen neuer Rows
- ☹️ Echte Aggregatfunktionen noch nicht möglich
- ☹️ Zum Teil noch widersprüchliche Dokumentation

trivadis
makes IT easier.

http://blog.sqlora.com/en/category/oracle/sql/ptf/

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

trivadis
makes IT easier.

Trivadis @ DOAG 2018 #opencompany

- Booth: 3rd Floor – next to the escalator
- We share our Know how!
  Just come across, Live-Presentations and documents archive
- T-Shirts, Contest and much more
- We look forward to your visit