PD ISO/IEC TR 19075-7:2017

BSI Standards Publication

Information technology — Database languages — SQL Technical Reports

Part 7: Polymorphic table functions in SQL

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

National foreword

This Published Document is the UK implementation of ISO/IEC TR 19075-7:2017.

The UK participation in its preparation was entrusted to Technical Committee IST/40, Data management and interchange.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

© The British Standards Institution 2017.
Published by BSI Standards Limited 2017

ISBN 978 0 580 95286 9
ICS 35.060

Compliance with a British Standard cannot confer immunity from legal obligations.

This Published Document was published under the authority of the Standards Policy and Strategy Committee on 30 April 2017.

Amendments/corrigenda issued since publication

Date	Text affected

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

TECHNICAL REPORT

ISO/IEC TR 19075-7

First edition
2017-03

Information technology — Database languages — SQL Technical Reports —

Part 7: Polymorphic table functions in SQL

Technologies de l'information — Langages de base de données — SQL rapport techniques —

Partie 7: Fonctions de table polymorphes dans SQL

Reference number
ISO/IEC TR 19075-7:2017(E)

Documents
Document Sharing Hub

© ISO/IEC 2017

Full document details: PD ISO/IEC TR 19075-7:2017

Contents

Page

Foreword . . . xi
Introduction . . . xii
1 Scope . . . 1
2 Normative references . . . 3
2.1 ISO and IEC standards . . . 3
3 Introduction to Polymorphic Table Functions . . . 5
3.1 Audiences . . . 5
3.2 Motivating examples . . . 6
3.2.1 CSVreader . . . 6
3.2.2 Pivot . . . 7
3.2.3 Score . . . 9
3.2.4 TopNplus . . . 12
3.2.5 ExecR . . . 15
3.2.6 Similarity . . . 16
3.2.7 UDjoin . . . 18
3.2.8 MapReduce . . . 19
3.3 The life cycle of a PTF . . . 20
4 PTF processing model . . . 23
4.1 Processing phases . . . 23
4.2 Virtual processors . . . 23
4.3 PTF component procedures . . . 23
4.4 Input table characteristics . . . 24
4.5 Partitioning and ordering . . . 25
4.6 Flow of control . . . 26
4.7 Flow of information . . . 27
4.8 Flow of row types . . . 28
4.9 Pass-through columns . . . 30
4.10 Security model . . . 30
4.11 Conformance features . . . 31
5 Specification . . . 35
5.1 Functional specification . . . 35
5.1.1 Parameter list . . . 35
5.1.2 Input table semantics . . . 36
5.1.3 Prunability . . . 37
5.1.4 Pass-through columns . . . 37

©ISO/IEC 2017 – All rights reserved

Documents Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

5.1.5 Result row type . . . 38
5.1.6 Determinism . . . 38
5.1.7 SQL-data access . . . 39
5.1.8 Documenting the PTF to the query author . . . 39
5.2 Design specification . . . 40
5.2.1 Name the component procedures . . . 40
5.2.2 Private data . . . 40
5.2.3 Routine characteristics of the component procedures . . . 41
5.2.4 Component procedure signatures . . . 42
6 Data definition language . . . 47
6.1 PTF creation . . . 47
6.2 PTF component procedures . . . 49
6.3 Altering PTF component procedures and PTFs . . . 49
6.4 Dropping a PTF and its component procedures . . . 50
7 Implementation . . . 51
7.1 PTF descriptor areas . . . 51
7.1.1 PTF descriptor area header . . . 52
7.1.2 SQL item descriptor areas for row types . . . 53
7.1.3 SQL item descriptor areas for partitioning . . . 57
7.1.4 SQL item descriptor areas for ordering . . . 57
7.2 PTF extended names . . . 57
7.3 Reading a PTF descriptor area . . . 58
7.4 Writing a PTF descriptor area . . . 58
7.4.1 Using DESCRIBE to populate a PTF descriptor area . . . 59
7.4.2 Using SET DESCRIPTOR to populate a PTF descriptor area . . . 59
7.4.3 Using COPY DESCRIPTOR to populate a PTF descriptor area . . . 60
7.5 Reading a PTF input cursor . . . 61
7.6 Outputting a row . . . 62
8 Invocation . . . 65
8.1 <table primary> . . . 65
8.2 <PTF derived table> . . . 65
8.3 Proper result correlation name and proper result column naming . . . 65
8.4 <routine invocation> . . . 66
8.5 <table argument> . . . 67
8.6 <table argument proper> . . . 68
8.6.1 <table or query name> . . . 68
8.6.2 <table subquery> . . . 68
8.6.3 Nested table function invocation . . . 69
8.7 Table argument correlation name . . . 69
8.8 Table argument column renaming . . . 70
8.9 Range variables and column renaming in nested PTF . . . 70
8.10 Partitioning . . . 70
8.11 Pruning . . . 71

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

8.12 Ordering . . . 71
8.13 Copartitioning . . . 72
8.14 Cross products of partitions . . . 73
8.15 <descriptor ar gument> . . . 74
9 Compilation . . . 75
9.1 Calling the describe component procedure . . . 75
9.2 Inside the describe component procedure . . . 75
9.3 Using the result of describe . . . 76
10 Optimization . . . 77
11 Execution . . . 79
11.1 Partitions and virtual processors . . . 79
11.2 Calling the start component procedure . . . 80
11.3 Inside the start component procedure . . . 81
11.4 Calling the PTF fulfill component procedure . . . 81
11.5 Inside the PTF fulfill component procedure . . . 81
11.6 Closing cursors . . . 81
11.7 Calling the PTF finish component procedure . . . 81
11.8 Inside the PTF finish component procedure . . . 82
11.9 Collecting the output . . . 82
11.10 Cleanup on a virtual processor . . . 82
11.11 Final result . . . 82
12 Examples . . . 83
12.1 Projection . . . 84
12.1.1 Overview . . . 84
12.1.2 Functional specification of Projection . . . 84
12.1.3 Design specification for Projection . . . 85
12.1.4 Projection component procedures . . . 85
12.1.5 Invoking Projection . . . 87
12.1.6 Calling Projection_describe . . . 87
12.1.7 Inside Projection_describe . . . 89
12.1.8 Result of Projection_describe . . . 91
12.1.9 Virtual processors for Projection . . . 91
12.1.10 Calling Projection_fulfill . . . 92
12.1.11 Inside Projection_fulfill . . . 93
12.1.12 Collecting the results . . . 93
12.1.13 Cleanup . . . 94
12.2 CSVreader . . . 95
12.2.1 Overview . . . 95
12.2.2 Functional specification of CSVreader . . . 95
12.2.3 Design specification for CSVreader . . . 95
12.2.4 CSVreader component procedures . . . 96
12.2.5 Implementation of CSVreader . . . 97
12.2.6 Invoking CSVreader . . . 98

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

12.2.7 Calling CSVreader_describe . . . 98
12.2.8 Inside CSVreader_describe . . . 100
12.2.9 Result of CSVreader_describe . . . 101
12.2.10 Virtual processor for CSVreader . . . 103
12.2.11 Calling CSVreader_start . . . 103
12.2.12 Inside CSVreader_start . . . 104
12.2.13 Calling CSVreader_fulfill . . . 104
12.2.14 Inside CSVreader_fulfill . . . 104
12.2.15 Collecting the output . . . 105
12.2.16 Calling CSVreader_finish . . . 105
12.2.17 Inside CSVreader_finish . . . 106
12.2.18 Cleanup . . . 106
12.3 Pivot . . . 107
12.3.1 Overview . . . 107
12.3.2 Functional specification of Pivot . . . 107
12.3.3 Design specification for Pivot . . . 108
12.3.4 Pivot component procedures . . . 108
12.3.5 Invoking pivot . . . 109
12.3.6 Calling Pivot_describe . . . 109
12.3.7 Inside Pivot_describe . . . 113
12.3.8 Result of Pivot_describe . . . 115
12.3.9 Virtual processors for Pivot . . . 117
12.3.10 Calling Pivot_fulfill . . . 117
12.3.11 Inside Pivot_fulfill . . . 118
12.3.12 Collecting the results . . . 118
12.3.13 Cleanup . . . 119
12.4 Score . . . 120
12.4.1 Overview . . . 120
12.4.2 Functional specification of Score . . . 120
12.4.3 Design specification for Score . . . 120
12.4.4 Score component procedures . . . 121
12.4.5 Invoking Score . . . 122
12.4.6 Calling Score_describe . . . 122
12.4.7 Inside Score_describe . . . 125
12.4.8 Result of Score_describe . . . 126
12.4.9 Virtual processors for Score . . . 127
12.4.10 Calling Score_fulfill . . . 129
12.4.11 Inside Score_fulfill . . . 129
12.4.12 Collecting the output . . . 130
12.4.13 Cleanup . . . 131
12.5 TopNplus . . . 132
12.5.1 Overview . . . 132
12.5.2 Functional specification of TopNplus . . . 132
12.5.3 Design specification for TopNplus . . . 132

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

12.5.4 TopNplus component procedures . . . 133
12.5.5 Invoking TopNplus. . . . 134
12.5.6 Calling TopNplus_describe. . . . 134
12.5.7 Inside TopNplus_describe. . . . 137
12.5.8 Result of TopNplus_describe. . . . 139
12.5.9 Virtual processors for TopNplus. . . . 139
12.5.10 Calling TopNplus_fulfill. . . . 141
12.5.11 Inside TopNplus_fulfill. . . . 141
12.5.12 Collecting the output. . . . 142
12.5.13 Cleanup. . . . 142
12.5.14 TopNplus using pass-through columns . . . 143
12.6 ExecR. . . . 145
12.6.1 Overview. . . . 145
12.6.2 Functional specification of Ex ecR. . . . 145
12.6.3 Design specification for Ex ecR. . . . 145
12.6.4 ExecR component procedures. . . . 146
12.6.5 Invoking ExecR. . . . 147
12.6.6 Calling Ex ecR_describe. . . . 147
12.6.7 Inside Ex ecR_describe. . . . 149
12.6.8 Result of Ex ecR_describe. . . . 150
12.6.9 Virtual processors for Ex ecR. . . . 150
12.6.10 Calling Ex ecR_start. . . . 151
12.6.11 Inside Ex ecR_start. . . . 152
12.6.12 Calling Ex ecR_fulfill. . . . 152
12.6.13 Inside Ex ecR_fulfill. . . . 152
12.6.14 Collecting the output. . . . 153
12.6.15 Calling Ex ecR_finish. . . . 153
12.6.16 Inside Ex ecR_finish. . . . 153
12.6.17 Cleanup. . . . 153
12.7 Similarity. . . . 154
12.7.1 Overview. . . . 154
12.7.2 Functional specification of Similarity. . . . 154
12.7.3 Design specification for Similarity. . . . 154
12.7.4 Similarity component procedures . . . 155
12.7.5 Invoking Similarity . . . 156
12.7.6 Calling Similarity_describe . . . 156
12.7.7 Inside Similarity_describe . . . 158
12.7.8 Result of Similarity_describe . . . 159
12.7.9 Virtual processors for Similarity. . . . 159
12.7.10 Calling Similarity_fulfill. . . . 163
12.7.11 Inside Similarity_fulfill. . . . 163
12.7.12 Collecting the output. . . . 164
12.7.13 Cleanup. . . . 164
12.8 UDjoin. . . . 165

©ISO/IEC 2017 – All rights reserved

Documents Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

12.8.1 Overview . . . 165
12.8.2 Functional specification of UDjoin . . . 165
12.8.3 Design specification for UDjoin . . . 165
12.8.4 UDjoin component procedures . . . 166
12.8.5 Invoking UDjoin . . . 167
12.8.6 Calling UDjoin_describe . . . 167
12.8.7 Inside UDjoin_describe . . . 168
12.8.8 Result of UDjoin_describe . . . 168
12.8.9 Virtual processors for UDjoin . . . 168
12.8.10 Calling UDjoin_fulfill . . . 168
12.8.11 Inside UDjoin_fulfill . . . 169
12.8.12 Collecting the output . . . 170
12.8.13 Cleanup . . . 170
12.9 Nested PTF in vocation . . . 171
12.9.1 Nested PTF syntax and semantics . . . 171
12.9.2 Nested PTF compilation . . . 174
12.9.3 Nested PTF execution . . . 175
12.9.4 The PTF author's view of nested PTF invocations . . . 176
Bibliography . . . 177
Index . . . 179

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

Tables

Table

Page

1 Primary audiences for Clauses and Subclauses in this Technical Report . . . 20
2 PTF routine characteristics . . . 41
3 Table parameter semantics . . . 43
4 Corresponding PTF component procedure parameters . . . 44
5 PTF descriptor area . . . 52
6 PTF descriptor area header . . . 52
7 Relevant SQL item descriptor components . . . 53

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

Figures

Figure

Page

1 PTF information flow 28
2 Row type relationships 29
3 Nested PTF data flow 172
4 Flow of row types 174
5 Simplified flow of row types 175
6 Net effect of complete compilation 175

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/IEC JTC 1, *Information technology,* Subcommittee SC 32 *Data management and interchange.*

A list of all parts in the ISO 19075 series can be found on the ISO website.

NOTE 1 — The individual parts of multi-part technical reports are not necessarily published together. New editions of one or more parts may be published without publication of new editions of other parts.

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

Introduction

The organization of this part of ISO/IEC 19075 is as follows:

1) Clause 1, "Scope", specifies the scope of this part of ISO/IEC 19075.

2) Clause 2, "Normative references", identifies additional standards that, through reference in this part of ISO/IEC 19075, constitute provisions of this part of ISO/IEC 19075.

3) Clause 3, "Introduction to Polymorphic Table Functions", provides an introduction to polymorphic table functions, the requirements leading to their incorporation into SQL, and illustrations of their use.

4) Clause 4, "PTF processing model", describes the abstract processing model for polymorphic table functions in the context of an SQL-implementation.

5) Clause 5, "Specification", describes the manner in which polymorphic table functions are specified in the SQL standard.

6) Clause 6, "Data definition language", provides the syntax and semantics of the SQL statements that create, modify, and drop polymorphic table functions.

7) Clause 7, "Implementation", guides authors of polymorphic table functions through the steps required to create all of the functions necessary to accomplish particular purposes.

8) Clause 8, "Invocation", supplies the information necessary for application writers, especially SQL query authors, to take advantage of the polymorphic table functions that are available to them.

9) Clause 9, "Compilation", is directed at the authors of polymorphic table functions and of SQL database systems to guide them in the steps required to compile polymorphic table functions in the context of a particular SQL-implementation.

10) Clause 10, "Optimization", describes the various aspects of polymorphic functions of which the authors of such functions and the authors of SQL-implementations must be aware to adequately optimize the execution of such functions.

11) Clause 11, "Execution", discusses the details of executing polymorphic table functions in the context of the processing model.

12) Clause 12, "Examples", supplies numerous examples in detail with commentaries to explain the various use cases, the requirements that relate to polymorphic table functions, and the specifics of the solutions for each use case.

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

Information technology — Database languages — SQL Technical Reports —

Part 7:
Polymorphic table functions in SQL

1 Scope

This Technical Report describes the definition and use of polymorphic table functions in SQL.

The Report discusses the following features of the SQL Language:

— The processing model of polymorphic table functions in the context of SQL.

— The creation and maintenance of polymorphic table functions.

— Issues related to methods of implementing polymorphic table functions.

— How polymorphic table functions are invoked by application programs.

— Issues concerning compilation, optimization, and execution of polymorphic table functions.

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

(Blank page)

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

2.1 ISO and IEC standards

[ISO9075-2] ISO/IEC 9075-2:2016, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*.

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

(Blank page)

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

3 Introduction to Polymorphic Table Functions

A polymorphic table function (abbreviated PTF) is a function that returns a table whose row type is not declared when the function is created. Rather, the row type of the result may depend on the function arguments in the invocation of a PTF, and therefore may vary depending on the precise syntax containing the PTF invocation. In addition, a PTF may have generic table parameters (*i.e.*, no row type declared when the PTF is created), and the row type of the result might depend on the row type(s) of the input tables. This Technical Report is intended to provide an informal description of PTFs, using examples and practical step-by-step advice on how to add a PTF capability to a relational DBMS, how to write a PTF, and how to invoke a PTF in an application.

3.1 Audiences

This Technical Report is written for three audiences:

1) The DBMS developer.

2) The PTF author.

3) The query author.

It is important to recognize that a PTF is somewhat like a view, only more procedural. With a view, there are the same three parties: DBMS, view author, and query author. The DBMS is the intermediary between the view author and the query author. The view is a way for the view author to "publish" an interface to tables without exposing the inner workings of the interface. Similarly, a PTF is a way for the PTF author to publish an interface to a procedural mechanism that defines a table. The query author only sees the published interface, whereas the DBMS and the PTF author share a more complex "private" interface. In particular, the query author sees a single PTF function, whereas the DBMS and the PTF author see a family of one to four related SQL-invoked procedures, called the PTF component procedures, and possibly additional private data.

Many sections of the Technical Report begin with a caption "Primary audience: xyz" stating which audience is most likely to find the section useful. However, anyone is welcome to read any section. Understanding the complete picture from all three perspectives is likely to be helpful to all three audiences.

A fourth audience can also be distinguished, the database administrator (DBA) as the one who is responsible for allocating disk storage for large tables. This is because disk allocation strategies can impact the performance of queries. The one allocating disk storage may be the query author or may be a separate role in the organization. If these are separate roles, the DBA may need to consult with the query author before the data is loaded. This is a complex issue because it may not be possible to anticipate the query at the time that the data is loaded, or the set of queries may be such that no one disk allocation scheme will allow optimal performance of all queries. The most relevant section for a DBA allocating disk storage is Subclause 11.1, "Partitions and virtual processors", where the audience is nominally the DBMS and the PTF author. However, specific advice about disk allocation is beyond the scope of this Technical Report.

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

3.2 Motivating examples

We begin by showing eight motivating examples that illustrate the capabilities of PTFs. These examples are presented from the standpoint of the query author, hiding the role of the PTF author and DBMS. The objective here is to get a taste of the power and generality of PTFs. The perspectives of the DBMS and the PTF author are explored at length in Clause 12, "Examples".

3.2.1 CSVreader

A spreadsheet can usually output a comma-separated list of values. Generally, the first line of the file contains a list of column names, and subsequent lines of the file contain data. The data in general can be treated as a large VARCHAR. However, some of the fields may be numeric or datetime.

The PTF author has provided a PTF called CSVreader designed to read a file of comma-separated values and interpret this file as a table. The query author can see this PTF in the Information Schema and knows that it has the following signature:

```
FUNCTION CSVreader (
    File VARCHAR(1000),
    Floats DESCRIPTOR DEFAULT NULL,
    Dates DESCRIPTOR DEFAULT NULL )
  RETURNS TABLE
  NOT DETERMINISTIC
  CONTAINS SQL
```

This signature has two parameter types that are distinctive to PTFs:

1) DESCRIPTOR is a type that is capable of describing a list of column names, and optionally for each column name, a data type. There is a helper function provided for the query author to construct a PTF descriptor area.

2) TABLE denotes the generic table type, a type whose value is a table. The row type of the table is not specified, and may vary depending on the invocation of the PTF.

In this example, the return type of CSVreader is TABLE. This is a distinguishing characteristic of every polymorphic table function: it returns a generic table.

The PTF author has published a user reference for CSVreader. The user reference tells the query author the semantics of the input parameters and what the output will be. In this example, the user reference documents the following:

1) The first parameter, File, is the name of a file on the query author's system. This file must contain the comma-separated values that are to be converted to a table. The first line of the file contains the names of the resulting columns. Succeeding lines contain the data. Each line after the first will result in one row of output, with column names as determined by the first line of the input.

2) Floats is a PTF descriptor area, which should provide a list of the column names that are to be interpreted numerically. These columns will be output with the data type FLOAT.

3) Dates is a PTF descriptor area, which provides a list of the column names that are to be interpreted as datetimes. These columns will be output with the data type DATE.

Based on the documentation in the user reference, the query author may write a query such as the following:

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

```
SELECT *
FROM TABLE ( CSVreader ( File => 'abc.csv',
                         Floats => DESCRIPTOR ("principle", "interest")
                         Dates => DESCRIPTOR ("due_date")
                       ) ) AS S
```

In the FROM clause, the TABLE operator introduces the invocation of a table function. A table function might be either a conventional (monomorphic) table function or a PTF. In this case, because CSVreader is declared with return type TABLE, this is a PTF invocation.

This invocation says that CSVreader should open the file called abc.csv. The list of output column names is found in the first line of the file. Among these column names, there must be columns named 'principle' and 'interest', which should be interpreted as numeric values, and a column named 'due_date', which should be interpreted as a date.

For example, suppose that the contents of abc.csv are

```
docno,name,due_date,principle,interest
123,Mary,01/01/2014,234.56,345.67
234,Edgar,01/01/2014,654.32,543.21
```

The result will be

docno	name	due_date	principle	interest
123	Mary	2014-01-01	234.56	345.67
234	Edgar	2014-01-01	654.32	543.21

The distinguishing feature of this example is that there are no input tables. Subsequent examples show various possibilities involving input tables.

This example is continued in detail in Subclause 12.2, “CSVreader”.

3.2.2 Pivot

In general, a pivot is an operation that reads a row and outputs several rows. Generally, the input is denormalized and the output is normalized. For example, maybe an input table has six columns, forming three pairs of (phone type, phone number), and the user wishes to normalize this into a table with two columns.

The PTF author has provided a PTF called Pivot; the query author can see the following signature in the Information Schema:

```
FUNCTION Pivot (
    Input TABLE PASS THROUGH WITH ROW SEMANTICS,
    Output_pivot_columns DESCRIPTOR,
    Input_pivot_columns1 DESCRIPTOR,
    Input_pivot_columns2 DESCRIPTOR DEFAULT NULL,
    Input_pivot_columns3 DESCRIPTOR DEFAULT NULL,
    Input_pivot_columns4 DESCRIPTOR DEFAULT NULL,
    Input_pivot_columns5 DESCRIPTOR DEFAULT NULL
```

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

```
) RETURNS TABLE
DETERMINISTIC
READS SQL DATA
```

The PTF author has provided a user reference that documents the following semantics for this PTF:

1) The first parameter, Input, is a generic table. This table is declared to have two options, "PASS THROUGH" and "WITH ROW SEMANTICS". These options have the following implications for the query author:

   a) WITH ROW SEMANTICS means that the result is determined on a row-by-row basis. The alternative, set semantics, will be seen in subsequent examples. At most one input table can have row semantics.

   b) PASS THROUGH means that, for each input row, the PTF makes the entire input row available in the output, qualified by a range variable associated with the input table. The alternative, NO PASS THROUGH, will be seen in some subsequent examples.

2) The second parameter, Output_pivot_columns, is a PTF descriptor area that lists the names of the columns that the query author wants to see in the result.

3) The third parameter, Input_pivot_columns1, is mandatory. This parameter is a PTF descriptor area that lists the names of the columns of the input table which are to be pivoted into the corresponding columns of the output table. There must be the same number of column names in the Output_pivot_columns PTF descriptor area and in the Input_pivot_columns1 descriptor area.

4) The remaining parameters, Input_pivot_columns2, Input_pivot_columns3, Input_pivot_columns4, and Input_pivot_columns5, are optional (indicated by the DEFAULT NULL declaration). If supplied, these are additional PTF descriptor areas for the input columns that are to be pivoted into the output columns. Each of these PTF descriptor areas must have the same number of column names as Output_pivot_columns.

This shows the capability to pivot at most 5 sets of columns. Of course, the PTF author could support many more by simply adding more optional parameters to the signature.

Based on this user documentation, the query author might write the following invocation:

```
SELECT D.Id, D.Name, P.Phonetype, P. Phonenumber
FROM TABLE ( Pivot ( Input => TABLE (Joe.Data) AS D,
                     Output_pivot_columns => DESCRIPTOR (phonetype, phonenumber),
                     Input_pivot_columns1 => DESCRIPTOR (phtype1, phonenumber1),
                     Input_pivot_columns2 => DESCRIPTOR (phtype2, phonenumber2)
                   ) ) AS P
```

In this invocation, the first TABLE () operator encloses the PTF invocation. The first parameter, called Input, passes a table, and uses the TABLE () operator to enclose the table name. This second TABLE () operator is required because SQL would normally interpret syntax such as Joe.Data as a column name rather than a table name. Since Joe.Data is a table, it is possible to assign it a correlation name, D in this example. (If an explicit correlation name is not provided, then the table name Joe.Data, or just Data, may be used as a range variable to reference it.) The remaining arguments are PTF descriptor areas, first of the output pivot columns and then the corresponding pairs of input pivot columns.

The query has two correlation names, D and P. D is associated with the input table Joe.Data whereas P is associated with the output of the PTF.

For input tables with pass-through columns, as in this example, the correlation name of the input table may be used as a qualifier to reference any column of the associated input table. (Input tables with set semantics follow a slightly different rule to be presented later.) In this example, D has been used to qualify the columns D.Id and D.Name.

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

P may be used to reference the columns that are produced by the PTF. In this example, P has been used to qualify the columns P.Phonetype and P.Phonenumber.

The result of the PTF invocation is a multiset of rows, each row having some columns qualified by D and some columns qualified by P. Every column of the input table Joe.Data is accessible in the columns qualified by D; the output columns of the PTF are referenceable using the correlation name P. In effect, each input row is concatenated with the columns that are produced by the PTF. This is a consequence of the fact that the input table has pass-through columns and row semantics: the result of the PTF is determined on a row-by-row basis (row semantics), and the entire input row is concatenated with the result of the PTF (pass-through columns). (The PTF may produce more than one row for a given input row; this will cause a "multiplier effect" in the output.)

For example, suppose Joe.Data has the following data:

ID	NAME	PHTYPE1	PHNUMBER1	PHTYPE2	PHNUMBER2
123	Mary	home	pqr	cell	stu
234	Edgar	home	vwx	work	xyz

The result will be

ID	NAME	PHONETYPE	PHONENUMBER
123	Mary	home	pqr
123	Mary	cell	stu
234	Edgar	home	vwx
234	Edgar	work	xyz

This example is continued in detail in Subclause 12.3, "Pivot".

3.2.3 Score

Score has two input tables:

1) One input table contains rows to be scored according to some algorithm.

2) The other input table (the model) contains the parameters for the algorithm that is used to score a row.

Each row can be scored independently of every other row. In contrast, every row of the model is required to specify the scoring algorithm.

The PTF author has provided a PTF called Score; the query author can see the following signature in the Information Schema:

```
FUNCTION Score (
    Data TABLE PASS THROUGH WITH ROW SEMANTICS,
```

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

```
    Model TABLE NO PASS THROUGH
      WITH SET SEMANTICS PRUNE WHEN EMPTY
) RETURNS TABLE (Score REAL)
DETERMINISTIC
READS SQL DATA
```

The first input table, called Data, contains the rows to be scored. Each row is scored independently of every other row, as indicated by WITH ROW SEMANTICS. The entire input row is accessible in the output, as indicated by PASS THROUGH.

The second input table, called Model, contains the parameters used for scoring a row. Since the entire data set is required to specify the algorithm, this table is declared as WITH SET SEMANTICS. A table with set semantics may be partitioned and/or ordered. Partitioning and ordering are decisions made by the query author, expressed in query syntax, as we shall see.

NO PASS THROUGH indicates that columns of the Model table are not copied to the output. However, if the input is partitioned, then the partitioning column(s) are still available in the output.

Since the algorithm cannot work with an empty model, the qualifier PRUNE WHEN EMPTY is added, indicating that the result of the PTF is empty if the model table is empty. This enables the DBMS to optimize by not even invoking the PTF when this table is empty.

The result, for each input row of Data, is a row concatenated from the following three sources:

1) The entire row of Data (because this has row semantics with pass-through columns).

2) The partitioning columns of Model, if any (because this has set semantics without pass-through columns).

3) An additional column named SCORE of type REAL containing the score for that row of Data.

The query author has a table containing a number of different models, which can be used to score rows for comparison against different "what if" scenarios. The query author writes the following query:

```
SELECT D.Id, D.S, D.T, M.Modelid, T.Score
FROM TABLE (Score ( Data =>  TABLE (MyData) AS D
                    Model => TABLE (Models) AS M
                                PARTITION BY Modelid
) AS T
```

This example has three correlation names, corresponding to the three sources for columns in the output rows:

1) D is the correlation name for the input table MyData. MyData has row semantics with pass-through columns and its correlation name D may be used to qualify any column of MyData.

2) M is the correlation name for the input table Models. Models has set semantics. It does not have pass-through columns but its correlation name M can be used to qualify the partitioning column Modelid.

3) T is the correlation name for the result of the PTF. T is used to qualify the additional column named SCORE.

This example introduces the PARTITION BY clause. Only tables with set semantics may be partitioned. The input table is partitioned as specified by the column(s) in the PARTITION BY clause; the PTF is evaluated independently on each partition. The SQL standard uses an abstraction called a virtual processor to specify the evaluation of a PTF. In this example, each partition is assigned to a separate virtual processor.

For example, perhaps Models contains the following rows:

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

Modelid	pname	pvalue
wet	x	19
wet	y	28
wet	z	37
dry	x	4
dry	y	5
dry	z	6

This tables contains two models, “wet” and “dry”, each having three parameters named “x”, “y”, and “z”, with parameter values in the column pvalue.

Table MyData may contain information to be scored by these two models:

id	s	t
122	9.4	3.4
233	8.4	6.5
344	10.2	9.3
455	11.0	8.8

The result might look like this:

id	s	t	Modelid	score
122	9.5	3.4	wet	12.9
233	8.4	6.5	wet	14.9
344	10.2	9.3	wet	19.5
455	11.0	8.8	wet	19.8
122	9.5	3.4	dry	6.4
233	8.4	6.5	dry	7.4
344	10.2	9.3	dry	9.2
455	11.0	8.8	dry	9.4

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

In the result, the first three columns are copied from MyData. The next column, M.Modelid, comes from the partitioning of Models. Every row of MyData is analyzed by both models, “wet” and “dry”. Since there are four rows in MyData and two models, there are eight rows in the result, four for each model. The last column is the score produced by the PTF.

This example is continued in detail in Subclause 12.4, “Score”.

3.2.4 TopNplus

TopNplus takes an input table that has been sorted on a numeric column. It copies the first *n* rows through to the output table. Any additional rows are summarized in a single output row in which the sort column has been summed and all other columns are null.

The query author sees the following signature in the Information Schema:

```
FUNCTION TopNplus (
    Input TABLE NO PASS THROUGH
          WITH SET SEMANTICS PRUNE WHEN EMPTY,
    Howmany INTEGER
  ) RETURNS TABLE
  NOT DETERMINISTIC
  READS SQL DATA
```

The PTF author has provided the following user documentation:

1) The first parameter, Input, is the input table. This table has set semantics, meaning that the result depends on the set of data (since the last row is a summary row). In addition, the table is marked as PRUNE WHEN EMPTY, meaning that the result is necessarily empty if the input is empty. The query author must order this input table on a single numeric column (syntax below).

2) The second parameter, Howmany, specifies how many input rows that the user wants to be copied into the output table; all rows after this will contribute to the final summary row in the output.

Using the user documentation, the query author might write the following query:

```
SELECT S.Region, T.*
FROM TABLE ( TopNplus ( Input => TABLE (My.Sales) AS S
                                  PARTITION BY Region
                                  ORDER BY Sales DESC,
                        Howmany => 3
                      )
           ) AS T
```

This example shows an input table that is both partitioned and ordered. In general, an input table with set semantics may be partitioned or ordered or both.

Consider the following input data representing the content of the table My.Sales:

Region	Product	Sales
East	A	1234.56
East	B	987.65

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

Region	Product	Sales
East	C	876.54
East	D	765.43
East	E	654.32
West	E	2345.67
West	D	2001.33
West	C	1357.99
West	B	975.35
West	A	864.22

The first five rows make up the partition with Region = 'East' and the last five rows make up the partition with Region = 'West'. Also notice that each partition has been sorted in descending order on Sales.

The DBMS creates two virtual processors, one for each partition. For example, on the virtual processor for Region = 'East', TopNplus sees the following input as S:

Region	Product	Sales
East	A	1234.56
East	B	987.65
East	C	876.54
East	D	765.43
East	E	654.32

In the other partition, for Region = 'West', TopNplus sees the following input as S:

Region	Product	Sales
West	E	2345.67
West	D	2001.33
West	C	1357.99
West	B	975.35
West	A	864.22

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

On each virtual processor, TopNplus copies the first 3 rows to the output (because Howmany = 3). However, it does not copy the partitioning column, since that is available to the query using the correlation name S.Then, TopNplus reads the remaining rows and computes the sum. In partition Region = 'East', the sum is 1419.75; in the other partition, the sum is 1839.57.

The result of the PTF invocation is shown below:

S	T	
Region	**Product**	**Sales**
East	A	1234.56
East	B	987.65
East	C	876.54
East		1419.75
West	E	2345.67
West	D	2001.33
West	C	1357.99
West		1839.57

Note that the result uses two correlation names: S, to qualify the partitioning column, and T, to qualify the columns that were output by TopNplus.

This example has been designed to show how the PTF can copy rows of input to the output without using Feature B205, “Pass-through columns”, which is an optional feature and may not be available in every implementation of polymorphic table functions. The example can also be modified a little to exploit pass-through columns if Feature B205, “Pass-through columns” is available. The modification is necessary because pass-through columns are an “all or nothing” capability — either an entire input row is copied to the output, or a row of nulls (except for the partitioning columns). In the results above, the first three rows in each partition can be copied to the output, where they would be qualified by S rather than T. The summary row, on the other hand, is not copied from any input row; therefore, the summary row would be null in the S.Product and S.Sales columns. To report the summary statistic, the PTF would use a separate column, qualified byT. Thus the result might look like this:

S			T
Region	**Product**	**Sales**	**Sales**
East	A	1234.56	1234.56
East	B	987.65	987.65
East	C	876.54	876.54

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

S			T
Region	**Product**	**Sales**	**Sales**
East			1419.75
West	E	2345.67	2345.67
West	D	2001.33	2001.33
West	C	1357.99	1357.99
West			1839.57

This example is continued in detail in Subclause 12.5, "TopNplus".

3.2.5 ExecR

R is a programming language used for analytic calculations. ExecR executes an R script on an input table. The PTF receives the R script as an input character string, but lacks the sophistication to analyze this R script to determine the row type of the result. Consquently, the query author bears the burden of specifying the output row type.

The query author sees the following signature in the Information Schema:

```
FUNCTION ExecR (
    Script VARCHAR(10000),
    Input TABLE NO PASS THROUGH
          WITH SET SEMANTICS KEEP WHEN EMPTY,
    Rowtype DESCRIPTOR )
  RETURNS TABLE
  NOT DETERMINISTIC
  READS SQL DATA
```

The PTF author supplies the following user documentation:

1) The first argument, Script, is a character string containing the R script to be executed.

2) The second argument, Input, is a generic input table with set semantics. This input table will be passed to the R script using an interface defined by the PTF author. The R script is expected to process the input table and produce an output table. Since it is possible that the R script might produce an output even if there is empty input, this generic table is marked KEEP WHEN EMPTY. Since the R script will not have the ability to copy input rows into output rows, the input table is NO PASS THROUGH.

3) The third argument, Rowtype, is a PTF descriptor area of the row type that the R script will produce.

Based on the PTF documentation, the query author might write this query:

```
SELECT D.Region, R.Name, R.Value
FROM TABLE (ExecR ( Script => '...',
                    Input => TABLE (My.Data) AS D
                               PARTITION BY Region,
```

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

```
                    Rowtype =>
                      DESCRIPTOR (Name VARCHAR(100), Value REAL)
                  )
            ) AS R
```

In this invocation, the things to note are:

1) The first argument is the R script to be evaluated.

2) The second argument is an input table with set semantics, partitioned by Region. In this example, the input is not sorted.

3) The third argument, Rowtype, provides a PTF descriptor area of two columns, called Name and Value, with types VARCHAR(100) and REAL, respectively.

The result rows are concatenated from two sources:

1) The partitioning column D.Region, qualified by the correlation name for the partitioned input table.

2) The columns R.Name and R.Value that are output by the PTF, qualified by R, the correlation name associated with the PTF.

This result is supported as follows:

1) The input data is partitioned on Region.

2) Each region effectively constitutes an independent data set.

3) The DBMS creates a virtual processor for each partition.

4) The virtual processor reads the data in a partition and produces a row with two columns, Name and Value.

5) The Region code is constant for all input rows on a virtual processor, and so the DBMS is able to augment the result from the PTF with the partition column.

6) In the SELECT list, the partitioning column is referenced using the correlation name D declared in the PARTITION clause, whereas the output of the PTF is referenced using the correlation name R.

Refer to Subclause 3.2.4, “TopNplus”, for a fuller example of partitioning, including an example of how the correlation names and result column names work.

The ExecR example is continued in detail in Subclause 12.6, “ExecR”.

3.2.6 Similarity

Similarity performs an analysis on two data sets, which are both tables of two columns, treated as the x and y axes of a graph. The analysis results in a number which indicates the degree of similarity between the two graphs, with 1 being perfectly identical and 0 being completely dissimilar. The numeric result is returned in a table with one row and one column. The result column is called Val and is of type REAL.

The query author sees the following signature in the Information Schema:

```
FUNCTION Similarlity (
    Input1 TABLE NO PASS THROUGH
             WITH SET SEMANTICS KEEP WHEN EMPTY,
    Input2 TABLE NO PASS THROUGH
```


©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

```
            WITH SET SEMANTICS KEEP WHEN EMPTY )
  RETURNS TABLE (Val REAL)
  NOT DETERMINISTIC
  READS SQL DATA
```

Note that in this example the result row type is known when creating the PTF; therefore, it can be specified in the DDL. The PTF author supplies the following documentation:

1) The two parameters are generic tables with set semantics. Each input table must be sorted on two numeric columns; these columns are interpreted as providing the points in an x-y plot.

2) Similarity performs an analysis on two data sets, resulting in a number which indicates the degree of similarity between the two graphs, with 1 being perfectly identical and 0 being completely dissimilar. The numeric result is returned in a table with one row and one column. The result column is called Val and is of type REAL.

The query author might write a query such as the following:

```
SELECT T1.Country, T2.Code, S.Val
FROM TABLE ( Similarity ( Input1 => TABLE (Sales) AS T1
                                    PARTITION BY Country
                                    ORDER BY (Qtr, Revenue),
                          Input2 => TABLE (Countries) AS T2
                                    PARTITION BY Code
                                    ORDER BY (Quarter, GDP)
                                    COPARTITION (T1, T2)
                          )
           ) AS S
```

This example has two partitioned input tables. When there is more than one partitioned input table, then Feature B202, "PTF Copartitioning" is relevant. If the SQL-implementation supports this feature, then the query syntax supports an optional COPARTITION clause. The COPARTITION clause specifies that the input tables identified by the correlation names T1 and T2 are to be copartitioned. Each partitioning list must have the same number of columns, and corresponding column names must be the comparable. In this example, the length of each partitioning list is 1, and the corresponding columns T1.Country and T2.Code must be comparable.

In execution, copartitioning works like this: The DBMS effectively forms a master list of all country codes from S3 and T3, eliminating duplicates. One way to do this is to perform this full outer equijoin:

```
SELECT *
FROM ( SELECT Country, 1 AS One
       FROM Sales ) AS S3
     FULL OUTER JOIN
     ( SELECT Code, 1 AS One
       FROM Countries ) AS T3
     ON ( S3.Country IS NOT DISTINCT FROM T3.Code )
```

(The IS NOT DISTINCT FROM predicate is *True* if the two comparands are equal or both null.)

For example, suppose that the distinct values of Sales.Country are 'CAN', 'JPN', and 'USA', whereas the distinct values of Countries.Code are 'CAN', 'JPN', and 'GBR'. The result of the preceding query is

S3.Country	S3.One	T3.Code	T3.One
CAN	1	CAN	1

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

S3.Country	S3.One	T3.Code	T3.One
JPN	1	JPN	1
USA	1		
		GBR	1

Thus there are four copartitions (for 'CAN', 'JPN', 'USA', and 'GBR') and the DBMS must start a virtual processor for each of them.

The result of the PTF invocation therefore has three columns:

1) The copartitioning column called Country.

2) The copartitioning column called Code.

3) The result of the PTF itself, called Val.

This example is continued in detail in Subclause 12.7, “Similarity”.

3.2.7 UDjoin

UDjoin performs a user-defined join. It takes two input tables, *T1* and *T2*, and matches rows according to some join criterion. It is intended that *T2* is ordered on a timestamp. UDjoin will analyze this ordered data into “clusters” of related rows, where each cluster is interpreted as representing some “event”. If two rows are tied in the ordering, they are placed in the same cluster. Some rows may be interpreted as “noise”, not representing any event.

After analyzing *T2* into event clusters, rows from *T1* are matched to the most relevant event cluster. It is possible that some rows of *T1* have no matching event cluster. It is also possible that some event clusters have no match in *T1*.

The output resembles a full outer join. If a row *R* of *T1* matches an event cluster *EC* of *T2*, then in the output *R* is joined to every row of *EC*. If *R* has no matching event cluster, then *R* is output with a null-extended row in place of the event cluster. Conversely, if an event cluster *EC* is not matched, then every row of *EC* is output with nulls in the portion of the output corresponding to *T1*.

Like a full outer join, there are range variables associated with each input table, which qualify output columns that correspond to columns of the input.

The PTF author creates this PTF with the following signature:

```
CREATE FUNCTION UDjoin (
  Candidates TABLE PASS THROUGH
    WITH SET SEMANTICS KEEP WHEN EMPTY,
  EventStream TABLE PASS THROUGH
    WITH SET SEMANTICS KEEP WHEN EMPTY
) RETURNS ONLY PASS THROUGH
```

The RETURNS ONLY PASS THROUGH syntax declares that the PTF does not generate any columns of its own; instead, the only output columns are passed through from input columns.

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

The query author might write the following query:

```
SELECT G.*, S.*
FROM TABLE (UDjoin ( Candidates => TABLE (Goods) AS G,
                     EventStream => TABLE (TimeSeries) AS S
                       ORDER BY Tstamp ) )
```

Note that the PTF does not generate any columns; therefore, there is no correlation name for the PTF itself, only for the input tables.

The result might look like this:

G			S		
Gid	**Golly**	**Wiz**	**Tstamp**	**Color**	**Shape**
125	Molly	Oz	3	Crimson	Diamond
125	Molly	Oz	4	Aquamaroon	Star
125	Molly	Oz	5	Vermilion	Pentagon
			8	Violet	Crescent
			9	Purple	Circle
			10	Plum	Ellipse
126	Dolly	Narnia			

In the results, the row (G.Gid = 125, G.Golly = 'Molly', G.Wiz = 'Oz') is matched to an event of three rows with Tstamp = {3, 4, 5}. The next event, with Tstamp = {8, 9, 10}, has no match in G, so the columns of G are null. The final row (G.Gid = 126, G.Golly = 'Dolly', G.Wiz = 'Narnia') has no matching event in S, where the columns are null.

3.2.8 MapReduce

MapReduce is a data processing paradigm using two phases, called Map and Reduce. In the classic “word count” example of the MapReduce paradigm, the Map phase reads one or more input files. Each input file is parsed into words separated by delimiters. Map outputs a series of records, each record being a tuple comprising a word and a count.These records are then partitioned on word. In the Reduce phase, the counts in each partition are summed. The final result is a list of words appearing in any of the input files, with their counts.

Map can be implemented using a PTF that takes as input a list of files, producing an output table with two columns, word and count. Reduce can then be performed using conventional SQL grouping and the COUNT aggregate.

In more general terms, the MapReduce paradigm has two phases, Map and Reduce. The Map phase analyzes its input into some fixed format suitable for input to the Reduce phase. The data is partitioned and Reduce performs some analysis on the partitioned data, which might not be supported by an SQL aggregate. In this

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

general paradigm, both the Map and the Reduce phases can be implemented by PTFs. To get the complete paradigm, the query author will write a functional composition of the two phases, schematically:

```
SELECT ...
FROM TABLE (Reduce (Map (...))) AS MT
```

The invocation of Map nested within Reduce does not need aTABLE operator because Map is known to return a table.

This paper does not discuss the MapReduce paradigm further. However, Subclause 12.9, "Nested PTF invocation", provides examples of nested PTF invocations.

3.3 The life cycle of a PTF

This paper will weave among the three audiences in order to arrive at a comprehensive understanding of the whole. We assume that both the PTF author and the query author edit text files in which they build their SQL statements. Realistically, the development cycle will include many iterations by the PTF author, and the query author may iterate the design of the query. These iterations are not shown in the our examples.

You, the reader, may fill any or all of these roles. If you are a DBMS developer, you will of course write the implementation of the DBMS functionality; during development and testing you will also play the role of PTF author and query author. If you are a PTF author, then you will also fill the role of query author to test your PTF. When testing a PTF, it is recommended that you use separate SQL-schemas for the PTF implementation and the PTF test suite. If you are a query author, you may still find it useful to understand how the DBMS and the PTF body interoperate to deliver the result of the PTF.

The remaining sections of this Technical Report and their primary audiences are shown in Table 1, "Primary audiences for Clauses and Subclauses in this Technical Report":

Table 1 — Primary audiences for Clauses and Subclauses in this Technical Report

PTF Author	DBMS Developer	Query Author
Subclause 4.1, "Processing phases"		
Subclause 4.2, "Virtual processors"		
Subclause 4.3, "PTF component procedures"		
Subclause 4.4, "Input table characteristics"		
Subclause 4.5, "Partitioning and ordering"		
Subclause 4.6, "Flow of control"		
Subclause 4.7, "Flow of information"		
Subclause 4.8, "Flow of row types"		
Subclause 4.9, "Pass-through columns"		

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

PTF Author	DBMS Developer	Query Author
Subclause 4.10, “Security model”		
Subclause 4.11, “Conformance features”		
Clause 5, “Specification”		
Clause 6, “Data definition language”		
Clause 7, “Implementation”		
	Clause 8, “Invocation”	
	Subclause 9.1, “Calling the describe component procedure”	
Subclause 9.2, “Inside the describe component procedure”		
	Subclause 9.3, “Using the result of describe”	
Clause 10, “Optimization”		
	Subclause 11.1, “Partitions and virtual processors”	
	Subclause 11.2, “Calling the start component procedure”	
Subclause 11.3, “Inside the start component procedure”		
	Subclause 11.4, “Calling the PTF fulfill component procedure”	
Subclause 11.5, “Inside the PTF fulfill component procedure”		
	Subclause 11.6, “Closing cursors”	
	Subclause 11.7, “Calling the PTF finish component procedure”	
Subclause 11.8, “Inside the PTF finish component procedure”		
	Subclause 11.9, “Collecting the output”	
	Subclause 11.10, “Cleanup on a virtual processor”	

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

PTF Author	DBMS Developer	Query Author
	Subclause 11.11, “Final result”	

The examples are developed in Clause 12, “Examples”, using the same outline.

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

4 PTF processing model

All of this section is addressed to the DBMS developer and the PTF author. The query author needs to understand input table semantics (Subclause 4.4, "Input table characteristics") and partitioning (Subclause 4.5, "Partitioning and ordering").

4.1 Processing phases

Primary audience: DBMS developer and PTF author

Processing a PTF invocation is divided into two phases: compilation and execution. These phases correspond to PREPARE and EXECUTE in dynamic SQL. In static SQL, the query author is not aware of the phases, since compilation is followed immediately by execution. However, the DBMS and the PTF author are always aware of these phases.

4.2 Virtual processors

Primary audience: DBMS developer and PTF author

The execution phase is described using an abstraction called a virtual processor. A virtual processor is a processing unit capable of executing a sequential algorithm. A virtual processor might be an actual physical processor (with associated operating system, *etc*.). Using techniques such as multiprocessing, a single physical processor might host several virtual processors. Virtual processors may execute independently and concurrently, either on a single physical processor or distributed across multiple physical processors. There is no communication between virtual processors. The DBMS is responsible for collecting the output on each virtual processor; the union of the output from all virtual processors is the result of the PTF.

4.3 PTF component procedures

Primary audience: DBMS developer and PTF author

The query author perceives a single PTF. The PTF author and the DBMS perceive that the PTF is composed of one to four PTF component procedures, which are invoked at various points during compilation and execution, as follows:

1) PTF describe component procedure: called once during compilation. The primary task of the PTF describe component procedure is to determine the row type of the output table. It can also initialize private data that will be passed to subsequent PTF component procedures. (This component procedure is optional.)

2) PTF start component procedure: called once per virtual processor to perform any initialization that is not done by the DBMS or the PTF describe component procedure (This component procedure is optional.)

©ISO/IEC 2017 – All rights reserved

Documents Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

3) PTF fulfill component procedure: called once per virtual processor to deliver the output table by "piping" rows to the DBMS. (This component procedure is required.)

4) PTF finish component procedure: called once per virtual processor to perform any clean up not performed by the DBMS. (This component procedure is optional.)

Thus a polymorphic table function is actually an organized collection of SQL-invoked procedures.

Any input scalars that are not compile-time constants are passed to the PTF describe component procedure as null. If the non-null input scalars are insufficient for the PTF describe component procedure to determine the output row type, then it may return an error, which effectively becomes a syntax error. Thus, the PTF author may effectively impose syntax constraints on the the query author by returning an error from the describe component procedure if the syntax requirements are not met.

The run-time PTF component procedures are used to process the input table(s) and generate the output table.

4.4 Input table characteristics

Primary audience: DBMS developer, PTF author, and query author.

Input tables are classified by three characteristics:

1) Input tables have either row semantics or set semantics, as follows:

   a) Row semantics means that the the result of the PTF is decided on a row-by-row basis. As an extreme example, the DBMS could atomize the input table into individual rows, and send each single row to a different virtual processor.

   b) Set semantics means that the outcome of the function depends on how the data is partitioned. A partition may not be split across virtual processors, nor may a virtual processor handle more than one partition.

2) The second characteristic, which applies only to input tables with set semantics, is whether the PTF can generate a result row even if the input table is empty. If the PTF can generate a result row on empty input, the table is said to be "keep when empty", meaning that the DBMS must actually instantiate a virtual processor (or more than one virtual processor in the presence of other input tables). The alternative is called "prune when empty", meaning that the DBMS can prune virtual processors from the query plan if the input table is empty. (Tables with row semantics are always effectively "prune when empty", so this choice is not relevant to them.)

3) The third characteristic is whether the input table supports pass-through columns or not. Pass-through columns is a mechanism enabling the PTF to copy every column of an input row into columns of an output row.

In the examples, the table parameters have characteristics as shown in the following table:

Example	Table Parameter	Row or set semantics?	Keep or prune when empty?	Pass-through?
CSVreader	no table parameters			

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

Example	Table Parameter	Row or set semantics?	Keep or prune when empty?	Pass-through?
Pivot	Input	row	(N/A)	yes
Score	Data	row	(N/A)	yes
	Model	set	prune	no
TopNplus	Input	set	keep	no (example could be worked using pass-through)
ExecR	Input	set	keep	no
Similarity	Input1	set	keep	no
	Input2	set	keep	no
UDjoin	Candidates	set	prune	yes
	EventStream	set	prune	yes

4.5 Partitioning and ordering

Primary audience: DBMS developer, PTF author, and query author.

The input tables with set semantics may be partitioned on one or more columns, at the discretion of the query author. An input table with at least one partitioning column is said to be partitioned (even if, in fact, the partitioning column has only one value). An input table with no partitioning column is said to be broadcast. In the examples, TopNplus, ExecR, and Similarity illustrate partitioning.

The input tables with set semantics may be ordered on one or more columns (other than partitioning columns — there is no benefit to ordering on a partitioning column), also at the discretion of the query author. The PTF is aware of the ordering during all PTF component procedures, and may utilize the ordering semantically (TopNplus illustrates this possibility). If the PTF does not use the ordering semantically, then there is no benefit to the query author in ordering the input table. Ordering an input table does not imply an ordering to the results or the query as a whole.

Input tables with row semantics may not be partitioned or ordered. Row semantics implies that the result can be determined on a row-by-row basis; therefore, the DBMS can assign rows of such tables to virtual processors arbitrarily, for example, using random, round robin, or load balancing algorithms. The Pivot and Score examples illustrate input tables with row semantics.

When there is more than one partitioned input table, they can, at the option of the query author, be copartitioned. With copartitioning, the copartitioned table arguments must have the same number of partitioning columns, and corresponding partitioning columns must be comparable. The DBMS effectively performs a full outer equijoin on the copartitioning columns, assigning one virtual processor to each combination of partitions

©ISO/IEC 2017 – All rights reserved

Documents Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

resulting from this equijoin. The Similarity example illustrates copartitioning. For detailed examples of copartitioning, see Subclause 12.7.9, "Virtual processors for Similarity".

4.6 Flow of control

Primary audience: DBMS developer and PTF author.

This section presents a conceptual architecture for the flow of control during the compilation and execution.

Refer to the following diagram, showing a schematic execution plan involving multiple virtual processors P_1, P_2, P_3, ...

Compile Time		**Virtual Processor**	**Run Time**		
Describe			**Start**	**Fulfill**	**Finish**
		P_1			
		P_2			
		P_3			

The flow of execution moves basically left to right. The describe step is not shown using any virtual processor, since it is an indivisible computational step.

The purpose of compilation is to set up for the subsequent execution phases. Compilation is performed without the ability to read the input data, though the row type and sort order of the input tables are passed to the PTF describe component procedure via PTF descriptor areas.

Compilation results in two kinds of information:

1) The row type of the result.

2) Values of the private variables of the PTF, if any. These values are saved by the DBMS and re-instantiated as input to the run-time PTF component procedures. This provides for information flow from the PTF describe component procedure to the run-time PTF component procedures.

The time between compilation and execution is indicated by the vertical blank space between them. If the query is prepared and executed as separate steps, there can be many executions, not portrayed in this diagram. For example, if the PTF invocation is in a view definition, then the PTF invocation is compiled when the view is defined and executed when the view is referenced in a query.

At run-time, the DBMS assembles the input data, partitions it, and directs each partition to a separate virtual processor. Each virtual processor executes independently of every other virtual processor. Virtual processors may be scheduled sequentially on the same physical processor, or concurrently on the same or different physical processors. Scheduling virtual processors is implementation-dependent.

Note that partitioning is only semantically correct if the overall task can be decomposed as a union of disjoint tasks. If an input table has row semantics, then the input table can be partitioned arbitrarily, so in that case the

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

partitioning decision is made by the DBMS according to an implementation-dependent algorithm. If an input table has set semantics, then the partitioning decision is the responsibility of the query author, and the default is to form a single partition.

4.7 Flow of information

Primary audience: DBMS developer and PTF author.

Information flows between stages of execution as follows:

1) The PTF describe component procedure receives a description of the input tables and their ordering (if any) as well as any scalar input arguments that are compile-time constants.

2) The PTF describe component procedure returns the row type of the PTF's result to the DBMS. This row type is also made available to all the run-time PTF component procedures. (If any table argument has pass-through columns, the handling of the result row type has some additional complexity, discussed in Subclause 4.9, “Pass-through columns”.)

3) The PTF describe component procedure might have private information to communicate to the other PTF component procedures that will execute in run time. For example, the PTF describe component procedure may go to some trouble to analyze the input scalars and tables, and it may be useful to pass a digest of this analysis to the run-time PTF component procedures, so that they do not need to repeat the analysis.

4) The run-time PTF component procedures may have information to pass from one stage to another. For example, if a resource is allocated during the start component procedure, then a handle for that resource should be passed to the fulfill component procedure to use the resource, and to the finish component procedure to deallocate it.

The following diagram illustrates the flow of information in a PTF invocation:

©ISO/IEC 2017 – All rights reserved

Documents Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

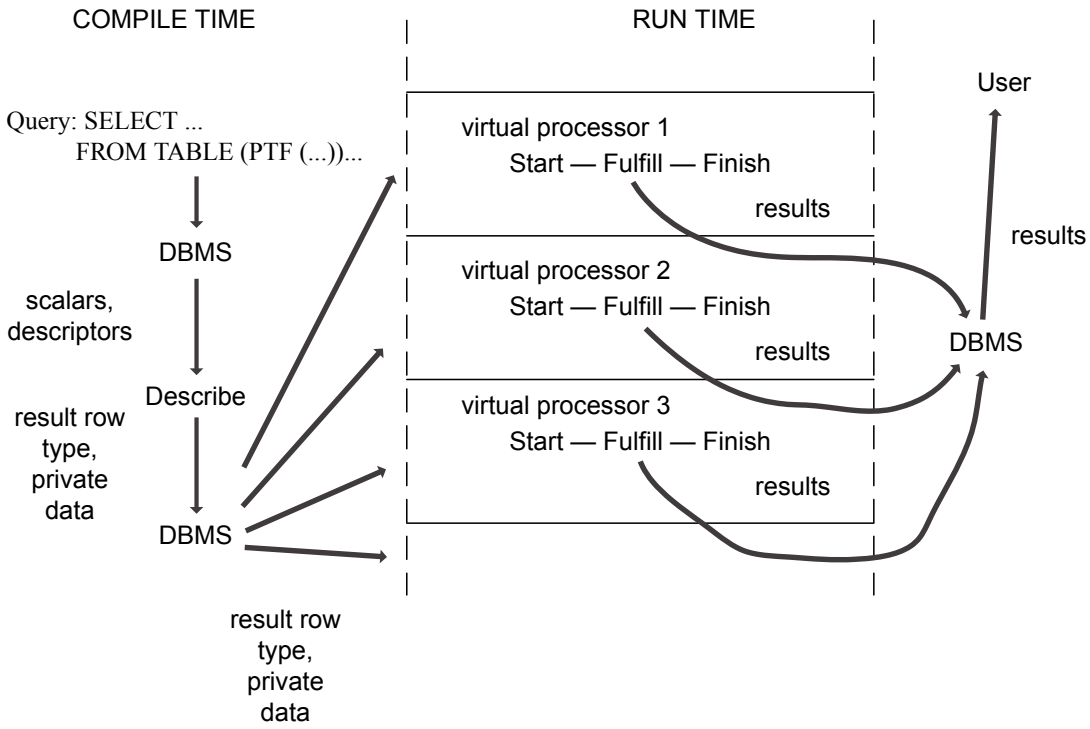

Figure 1 — PTF information flow

4.8 Flow of row types

Primary audience: DBMS developer and PTF author.

For each table argument, a PTF invocation has the following input row types:

1) The full row type of an input table, including every column of the input table. If the input table has column renaming (specified by a<parenthesized derived column list>), then the column names in the<parenthesized derived column list> are used; otherwise, the original column names of the input table are used.

2) The requested row type of an input table; this includes only those columns that the PTF describe component procedure requests to receive from the DBMS. The PTF describe component procedure is responsible for describing this row type; it must not be empty. (If the PTF describe component procedure is absent, then this is a copy of the full row type.)

3) The cursor row type of an input table.This is the same as the requested row type, plus one additional column (the pass-through input surrogate column) if the input table has pass-through columns (see Subclause 4.9, "Pass-through columns").

A PTF invocation also has the following result row types:

©ISO/IEC 2017 – All rights reserved
Documents Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

1) The initial result row type. This row type lists the columns that the PTF itself will generate (called the proper result columns of the PTF).

2) The intermediate result row type. This is identical to the initial result row type, plus one additional column (the pass-through output surrogate column) for each input table that has pass-through columns.This is the row type that must be used when performing <pipe row statement> during the execution phase to output a row.

3) The external result row type. This is the same as the initial result row type, except that the proper result columns may be renamed by the query using a <parenthesized derived column list>.

4) The complete result row type (called just the "row type of <table primary>" in the standard) comprising the external result row type plus, for each table argument *TA*:

   a) If *TA* has pass-through columns, then, for every column of *TA*, a result column having the same name and data type.

   b) Otherwise, for every partitioning column of *TA*, a result column having the same name and data type.

The relationships between these row types is illustrated in the following diagram:

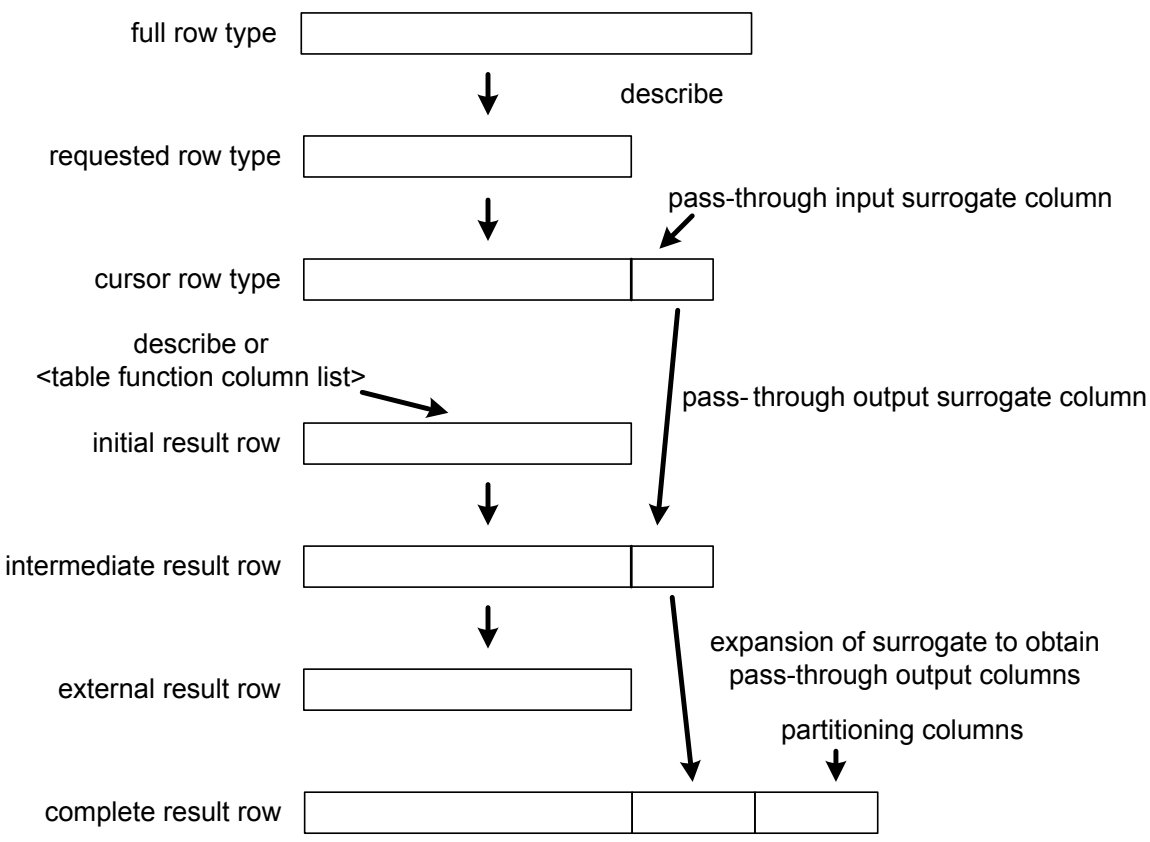

Figure 2 — Row type relationships

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

4.9 Pass-through columns

Primary audience: DBMS developer and PTF author.

Pass-through columns are a mechanism whereby a PTF can copy all columns of an input row into a result row, without needing to fetch the columns individually, and without needing to understand their data types. From the PTF's perspective, pass-through columns are condensed into a single surrogate value. The query is not aware of the surrogates and instead sees that an input row has been copied into the result.

Refer to Figure 2, "Row type relationships". For each table argument, typically the PTF describe component procedure requests a subset of the full row type, comprising just the columns that the PTF needs to examine for its semantics. If the input table has pass-through columns, then the cursor row type is formed by appending a single column of implementation-dependent name and type, called the pass-through input surrogate column.

As for the result of a PTF, either the PTF describe component procedure or the <table function column list> declared in the CREATE FUNCTION statement specifies the initial result row type, consisting of the proper result columns (the columns that the PTF itself will generate). The intermediate result row type consists of a copy of the initial result row type, plus one pass-through output surrogate column for each table argument that has pass-through columns.

During the execution phase, the PTF fulfill component procedure receives PTF descriptor areas for the cursor row types and the intermediate result row type. When the PTF fulfill component procedure fetches from a cursor, the DBMS populates the pass-through input surrogate column with an opaque value that represents the non-partitioning columns of the table, including both requested and non-requested columns. For example, this surrogate might be formed by compressing the columns into a BLOB, or it might be a candidate key. (Note that by using common column projection algorithms, the surrogate value actually only needs to represent those columns that are actually referenced later in the query.)

The PTF fulfill component procedure can copy the value of the pass-through input surrogate column to the pass-through output surrogate column in the intermediate result row. The intermediate result row is sent to the DBMS by a <pipe row statement>, whereupon the DBMS expands the surrogate value to reconstruct the non-partitioning columns that the surrogate value represents.

Alternatively, the PTF fulfill component procedure can place a null value in the pass-through output surrogate column. When transmitted to the DBMS by a<pipe row statement>, the null value of the surrogate will expand into null values in all the pass-through result columns. (Note that this has no effect on partitioning columns, since the surrogate does not represent them.) This is the only output scenario available to the PTF start and finish component procedures, which can use a <pipe row statement>, but have no cursor and hence no input surrogate value.

Partitioning columns are handled separately from pass-through columns. The values of partitioning columns are invariants on a virtual processor. Since the virtual processor is the processing unit that processes a <pipe row statement>, the virtual processor is able to add the values of the partitioning columns into the complete result row.

4.10 Security model

Primary audience: DBMS developer and PTF author.

The security model for PTFs is as follows:

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

1) The query author must have EXECUTE privilege on the PTF in order to invoke it.

2) The query author must have SELECT privilege on input tables (more precisely, on the columns of the input tables).

3) In this Technical Report, the owner of the PTF is generally portrayed as the owner of the PTF component procedures. This is likely to be the case in practice, but the minimal requirement is merely that the owner of the PTF has EXECUTE privilege on all PTF component privileges.

4) The PTF does not need SELECT privilege on the input tables or their columns; only the query author needs privilege on the input tables. The DBMS opens the input tables and passes open read-only cursors to the PTF fulfill component procedure. These cursors are anonymous in the sense that the PTF does not know the identity of the input tables. The only operation the PTF can perform on these input cursors is FETCH.

5) If a PTF needs a side table to perform a "table lookup", the PTF author has three ways to do this:

   a) If the lookup table is proprietary to the PTF (perhaps it is the intellectual property of the PTF author), then the PTF may perform SELECT operations on the proprietary table by opening it in the PTF component procedures using "definer's rights". There is no need to grant SELECT on the proprietary table to the query author.

   b) If the lookup table is not proprietary to the PTF, then the PTF can expect the query author to pass the lookup table as an input table, in which case the table will be subject to access checking using the query author's privileges.

   c) If the preceding techniques are not sufficient, then the PTF can expect the query author to pass text arguments containing the names of tables, *etc*., from which the PTF can build a dynamic query. This dynamic query must be access-checked using the query author's privileges, so the PTF component procedure that does this must be created with "invoker's rights".

Note that the implementation techniques described in Clause 7, "Implementation", have no access checking and can be used freely in either definer's rights or invoker's rights component procedures. The PTF author does not need to be concerned with definer's rights or invoker's rights unless the PTF falls under either scenario a) or c) above.

4.11 Conformance features

Support for polymorphic table functions is an optional feature of SQL. If the DBMS provides minimal support for polymorphic table functions, as specified in [ISO9075-2], then the DBMS can claim support for Feature B200, "Polymorphic table functions".

[ISO9075-2] specifies additional advanced features that require support for Feature B200, "Polymorphic table functions", and enrich that minimal support with extra functionality. These additional conformance features are as follows:

— Feature B201, "More than one PTF generic table parameter"

  This feature permits a polymorphic table function to have more than one generic table parameter. Examples of multi-table input to a PTF are found in Subclause 12.4, "Score", Subclause 12.7, "Similarity", and Subclause 12.8, "UDjoin".

— Feature B202, "PTF Copartitioning"

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

This feature provides support for copartitioning. The example for copartitioning is found in Subclause 12.7, "Similarity".

— Feature B203, "More than one copartition specification"

This feature permits a polymorphic table to have more than one copartitioning specification. At least four input tables are required to utilize this feature (two input tables for each of two copartitioning specifications). There are no examples of this feature in this Technical Report.

— Feature B204, "PRUNE WHEN EMPTY"

This feature permits the DBMS to avoid creating a virtual processor for a partition that is known to be empty. Syntactically, it can be specified by either the PTF author or the query author.

The PTF author specifies PRUNE WHEN EMPTY in DDL syntax of a table parameter if the PTF author knows that the result of the PTF for a partition is empty when the input partition has no rows. From the standpoint of the DBMS, it is an optimization if the DBMS can avoid creating a virtual processor for an empty input partition. However, from a functionality standpoint, the outcome is the same whether the virtual processor is created or not, since the result is empty in either case. If the PTF can generate a result even when given an empty input data set, the PTF author should not specify PRUNE WHEN EMPTY. Examples of this DDL syntax are found in Subclause 12.4, "Score", and Subclause 12.5, "TopNplus".

If the PTF can generate a result on an empty input partition, the query author may not be interested in that result. In that case the query author can specify PRUNE WHEN EMPTY in the query syntax. An example of PRUNE WHEN EMPTY in query syntax is found in Subclause 12.7, "Similarity".

— Feature B205, "Pass-through columns"

Pass-through columns are a device that the DBMS can provide to the PTF author, making it easy for the PTF author to copy an input row into the output. Examples of this are found in Subclause 12.1, "Projection", Subclause 12.3, "Pivot", Subclause 12.4, "Score", and Subclause 12.8, "UDjoin". In addition, Subclause 12.5, "TopNplus", shows how the PTF author can copy an input row to an output row even if the DBMS does not support this Feature.

— Feature B206, "PTF descriptor parameters"

PTF descriptor parameters are a mechanism for the query author to pass a row type as an argument to a polymorphic table function. Examples of PTF descriptor parameters are found in Subclause 12.1, "Projection", Subclause 12.2, "CSVreader", Subclause 12.3, "Pivot", and Subclause 12.6, "ExecR".

— Feature B207, "Cross products of partitionings"

With this feature, if an invocation of a polymorphic table function has more than one partitioned input table, then it is the query author's choice whether to relate the partitioned input tables using copartitioning. (Otherwise, all partitioned input tables must be related to one another via a single copartitioning specification.) The possibility of multiple partitioned input tables that are not copartitioned is illustrated in Subclause 12.7.9, "Virtual processors for Similarity", following the example of copartitioning.

There are also two conformance features that are relevant only to the PTF author and the DBMS developer:

— Feature B208, "PTF component procedure interface"

[ISO9075-2] specifies an optional interface between the DBMS and the polymorphic table function. The interface is provided as a specification device, to specify the semantics of an invocation of a polymorphic table function. An SQL-implementation is not required to use the specified interface; it may substitute an equivalent interface that provides the same functionality to the PTF author. If the DBMS adheres to the

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

interface as specified in [ISO9075-2], then the DBMS may claim conformance to Feature B208, “PTF component procedure interface”. All of the examples in this Technical Report assume this interface.

— Feature B209, “PTF extended names”

PTF extended names are a distinctive category of dynamic extended names, used to name PTF cursors and PTF descriptor areas. PTF extended names are part of the optional interface between the DBMS and the polymorphic table function. It is possible that an SQL-implementation may choose to support PTF extended names without supporting other aspects of the interface. In that case, the DBMS may claim conformance to Feature B209, “PTF extended names”, even if it does not conform to Feature B208, “PTF component procedure interface”. All of the examples in this Technical Report assume this feature.

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

(Blank page)

©ISO/IEC 2017 – All rights reserved
Documents
Document Sharing Hub
Full document details: PD ISO/IEC TR 19075-7:2017

5 Specification

Primary audience: PTF author

Specification is the planning phase of software development, prior to implementation. For our purpose in this Technical Report, the specification phase results in what we call “skeleton DDL”, that is, all of the CREATE statements necessary to define a PTF, except for the body of the PTF. Specification can be divided into two subphases:

— Functional specification specifies the software as seen from the outside (as a “black box”). This phase specifies the user interface and semantics of the software. The functional specification can become the basis for documentation addressed to the query author.

— Design specification specifies the software as seen from the inside (“white box”).This may be proprietary information not divulged to the query author.

5.1 Functional specification

The first step in the life cycle is to write a functional specification for the PTF. The functional specification will describe the user interface and semantics of the PTF, without describing the inner design of the PTF. The functional specification becomes the basis for documentation supplied to the query author.

5.1.1 Parameter list

The first step in writing a functional specification is to decide the parameter list, which might include the following things:

1) The input table(s). These are generic tables, so the input tables should be thought of in terms of their role within the transformation that the PTF implements. Note that Feature B201, “More than one PTF generic table parameter”, is required if the PTF has more than one input table.

2) The scalar inputs. The PTF can use scalar inputs to parameterize the behavior of the PTF.

3) The PTF descriptor area inputs. A PTF descriptor area can provide a list of column names, possibly augmented by data types. Of course, a list of column names can be provided via a character string scalar; however, this requires the PTF to provide a parsing capability, with attention to case-sensitivity rules. In general, if a PTF can deduce a list of columns without a PTF descriptor area, that will be preferable from the standpoint of the PTF author's customer, the query author. However, in totally dynamic situations, the query author may have to provide column lists, and PTF descriptor areas will probably be the most convenient way to do this. PTF descriptor areas are discussed in Subclause 7.1, “PTF descriptor areas”. The examples Pivot and ExecR illustrate descriptor parameters. Note that Feature B206, “PTF descriptor parameters”, is required if there are any PTF descriptor area inputs.

©ISO/IEC 2017 – All rights reserved

Documents Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

After deciding on the parameter list, the PTF author is ready to write the first skeleton CREATE FUNCTION statement. At this stage we have an incomplete CREATE FUNCTION because it only lists the input parameters (there is more DDL to come later). The parameters are declared with the following types:

— Input tables have parameter type TABLE.

— Input scalars have their usual parameter types (VARCHAR, INTEGER, *etc.*).

— Input PTF descriptor areas have parameter type DESCRIPTOR.

Thus, at this stage, the function declaration looks something like this:

```
CREATE FUNCTION ptf (
  Input TABLE,
  Scalar INTEGER,
  Columns DESCRIPTOR )
RETURNS TABLE
```

At this stage the PTF author may also be able to decide which inputs are optional. Optional inputs are indicated by specifying a default value; for example:

```
CREATE FUNCTION ptf (
  Input TABLE,
  Scalar INTEGER DEFAULT 3,
  Columns DESCRIPTOR DEFAULT NULL )
RETURNS TABLE
```

Input tables are always mandatory; you cannot specify a default for a table input.

Advice to the DBMS developer: you want to support the PTF author at this early stage. The CREATE FUNCTION statement above is incomplete and not suitable for actual use. Nevertheless, the DBMS may want to allow the PTF author to load such a definition as a kind of "invalid" function definition. This will facilitate a DBMS tool that can assist the PTF author as the latter goes through the stages of development outlined here. We will talk more about how the DBMS can assist the PTF author later in Subclause 5.2.4, "Component procedure signatures".

5.1.2 Input table semantics

After listing the parameters, the next specification step is to classify each input table as row semantics or set semantics.

— Row semantics means that the the result of the PTF is decided on a row-by-row basis. As an extreme example, the DBMS could atomize the input table into individual rows, and send each single row to a different virtual processor. Or the DBMS might process them all on the same virtual processor. A table should be given row semantics if the PTF does not care how rows are assigned to virtual processors.

— Set semantics means that the outcome of the function depends on how the data is partitioned. A table should be given set semantics if all rows of a partition should be processed on the same virtual processor. This is the default semantics.

At most one input table may have row semantics; all other input tables must have set semantics. A PTF that has an input table with row semantics is said to be a per-row PTF; otherwise, the PTF is said to be a per-set PTF.

©ISO/IEC 2017 – All rights reserved

Documents
Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017

In our examples:

— CSVreader has no input tables.

— Pivot has an input table with row semantics.

— Score has one input table with row semantics and one with set semantics.

— TopNplus has an input table with set semantics.

— ExecR has an input table with set semantics.

— Similarity and UDjoin have two input tables with set semantics.

5.1.3 Prunability

If the DBMS supports Feature B204, "PRUNE WHEN EMPTY", then prunability is the next step in functional specification. In this case, if an input table has set semantics, then there is a further property: whether the table can be pruned or not. An input table is prunable if the the result of the PTF with an empty input table is an empty output table. (Prunability is not a choice for input tables with row semantics, since an empty input row necessarily generates no output rows.)

We have five examples that have input tables with set semantics:

— Score has one table parameter with row semantics and one with set semantics. The second table parameter is used to define a model to score rows of the first parameter. It is impossible to set up a model with an empty table, so this is be PRUNE WHEN EMPTY.

— TopNplus: on an empty input, it would be very reasonable to make an empty output, so we could specify PRUNE WHEN EMPTY. Alternatively, we might want to generate a single row with a sum of 0 in the sort column and the other columns null. If we do that, then we should specify KEEP WHEN EMPTY.

— ExecR: An R script could potentially have output even on an empty input, therefore ExecR should be KEEP WHEN EMPTY

— Similarity: we'll assume that the similarity algorithm computes that an empty table is completely dissimilar from a non-empty table (result = 0), and completely similar to another empty table (result = 1). Thus, there is a result even if one or both inputs is empty; therefore, both input tables should be KEEP WHEN EMPTY.

— UDjoin: since the semantics resemble a full outer join, we may have a result even if either table is empty, so both input tables should be KEEP WHEN EMPTY.

If the DBMS does not support Feature B204, "PRUNE WHEN EMPTY", then every table is effectively KEEP WHEN EMPTY. This does not impact the functionality of the PTF (an empty partition simply generates an empty result), but it may cost some performance since the DBMS may needlessly instantiate virtual processors for empty partitions.

5.1.4 Pass-through columns

If the SQL-implementation supports Feature B205, "Pass-through columns", then the final characteristic of input tables is whether they support pass-through columns or not. Pass-through columns is a feature that enables

©ISO/IEC 2017 – All rights reserved

Documents Document Sharing Hub

Full document details: PD ISO/IEC TR 19075-7:2017