

*SDMX Technical Working Group
VTL Task Force*

Validation & Transformation Language

Part 2 - Library of Operators

Version 1.0

February 2015

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29

31 Foreword

32 The SDMX Technical Working Group is pleased to present the version 1.0 of the Validation
33 and Transformation Language, in short VTL.

34
35 The work on VTL was launched at the end of 2012 by the SDMX Secretariat. SDMX already has
36 a package for transformations and expressions which is present in the information model,
37 although a specific language does not exist yet. To make this framework fully operational, a
38 standard “language” for defining validation and transformation rules (set of operators, their
39 syntax and semantics) should be adopted, appropriate IT formats for exchanging such rules
40 and related metadata should be introduced, and the web services to store and retrieve them
41 should be designed.

42
43 A task force was put in place, composed of members of SDMX, DDI and GSIM communities and
44 the work started in summer 2013. The intention was to provide a language which is usable by
45 statisticians to express logical validation rules and transformations on data, whether
46 described as dimensional tables or as unit-record data. The assumption is that this logical
47 formalization of validation and transformation rules would be converted into specific
48 programming languages for execution (SAS, R, Java, SQL, etc.) but would provide a “neutral”
49 expression at business level of the processing taking place, against which various
50 implementations can be mapped. Experience with existing examples suggests that this goal
51 would be attainable.

52
53 An important point that emerged is that several standards are interested in such a language.
54 However, each standard operates on its model artefacts and produces artefacts within the
55 same model (property of closure). To cope with this, VTL has been built upon a very basic
56 information model, taking the common parts of GSIM, SDMX and DDI, mainly using artefacts
57 from GSIM 1.1, somewhat simplified and with some additional detail. This way the existing
58 standards (SDMX, DDI, others) may adopt VTL by mapping their information model against
59 the VTL one. Therefore, although a work-product of SDMX, the VTL language will be usable
60 also with other standards.

61
62 The VTL 1.0 package includes:

- 63 a) Part 1, highlighting the main characteristics of VTL, its core assumptions and the
64 information model the language is based on;
- 65 b) Part 2, containing the full library of operators ordered by category, including examples;
- 66 c) BNF notation (Backus-Naur Form) which is the technical notation to be used as a test bed
67 for all the examples throughout the document.

68
69 The present document (VTL part 2) contains the full list of VTL Operators and describes their
70 main characteristics. This first version can support validation and basic compilation needs.
71 Future versions will include more features related to transformation of data.

72 The latest version of the VTL is freely available online at www.sdmx.org.

73

74 **Acknowledgements**

75 This publication has been prepared thanks to the collective input of experts from Bank of
76 Italy, Bank for International Settlements (BIS), European Central Bank (ECB), Eurostat, ILO,
77 ISTAT (Italy), OECD, Statistics Netherlands, and UNESCO. Other experts from the SDMX
78 Technical Working Group, the SDMX Statistical Working Group and the DDI initiative were
79 consulted and participated in reviewing the documentation.

80 The list of authors, contributors and reviewers includes the following experts: Sami Airo,
81 Foteini Andrikopoulou, David Barraclough, Luigi Bellomarini, Marc Bouffard, Angel Simon
82 Delgado, Vincenzo Del Vecchio, Fabio Di Giovanni, Jens Dossé, Heinrich Ehrmann, Bryan
83 Fitzpatrick, Arofan Gregory, Edgardo Greising, Angelo Linardi, Chris Nelson, Stratos
84 Nikoloutsos, Marco Pellegrino, Michele Romanelli, Juan Alberto Sanchez, Daniel Suranyi, Olav
85 ten Bosch, Laura Vignola, Nikolaos Zisimos.

86 Feedback and suggestions for improvement are encouraged and can be sent to the SDMX
87 Technical Working Group (twg@sdmx.org).

88

89 Table of contents

90		
91	FOREWORD.....	3
92	TABLE OF CONTENTS.....	5
93	VTL LIBRARY GENERAL PICTURE	7
94	DIAGRAM OF THE OPERATORS	7
95	LIST OF THE OPERATORS	8
96	EVALUATION ORDER OF THE OPERATORS	12
97	WRITING CONVENTIONS FOR OPERATORS AND TYPES	14
98	GENERAL PURPOSE OPERATORS	17
99	ROUND PARENTHESES ().....	17
100	ASSIGNMENT :=	17
101	MEMBERSHIP #.....	18
102	GET	19
103	PUT	25
104	EVAL.....	26
105	STRING OPERATORS.....	29
106	LENGTH.....	29
107	TRIM.....	32
108	UPPER/LOWER	34
109	SUBSTR	35
110	INDEXOF.....	37
111	MATHEMATICAL OPERATORS.....	40
112	UNARY PLUS +	40
113	UNARY MINUS -	41
114	ADDITION AND SUBTRACTION + ++ - --	43
115	MULTIPLICATION AND DIVISION * ** / //	47
116	ROUND	50
117	ABS	52
118	TRUNC	53
119	EXP.....	55
120	LN	56
121	LOG	58
122	POWER.....	60
123	NROOT.....	61
124	MOD.....	63
125	BOOLEAN OPERATORS	65
126	AND.....	65
127	OR.....	67
128	XOR.....	70
129	NOT.....	72
130	RELATIONAL OPERATIONS.....	75
131	UNION (RECORDS).....	75

132	INTERSECT	78
133	SYMDIFF	79
134	SETDIFF	81
135	MERGE	83
136	STATISTICAL	86
137	MIN/MAX	86
138	HIERARCHY	87
139	VALIDATION OPERATORS	99
140	EQUAL TO =	100
141	NOT EQUAL TO <>	101
142	GREATER THAN > >=	103
143	LESS THAN < <=	106
144	IN, NOT IN	108
145	BETWEEN	109
146	ISNULL	113
147	EXISTS_IN, NOT_EXISTS_IN/IN_ALL	115
148	CHECK	119
149	MATCH_CHARACTERS	123
150	MATCH_VALUES	125
151	CONDITIONAL OPERATOR	127
152	IF-THEN-ELSE	127
153	NVL	130
154	CLAUSES (POSTFIX OPERATORS)	133
155	RENAME	133
156	FILTER	134
157	KEEP	135
158	CALC	137
159	ATTRCALC	139
160	AGGREGATE	142
161	USE CASES	150
162	LABOUR FORCE SURVEY AND VTL	150
163		

164 VTL library general picture

165 The VTL library of the Operators is described hereinafter. This is the first version of the
166 library, oriented mainly to data validation purposes. The operators included in this version of
167 the VTL are summarized in the diagrams and tables below.

168 Diagram of the Operators



169
170
171

172 List of the Operators

173 The following table lists the VTL Operators and describes their main characteristics. They are
 174 ordered by category except the clauses, which are the operators having a postfix syntax, that
 175 are shown all together in the end.

176 VTL includes operators that may operate both on Data Sets and on Structure Components of
 177 the Data Sets. The last column shows if the Operator has a version that operates on
 178 Components (besides the version that operates on Data Sets) or if it operates only on Data
 179 Sets. The Component version takes as input and returns in output Component expressions.
 180 They are not typically used in assignments, since Components cannot be stored directly into
 181 the system. Instead, they are part of the syntax of other operators or clauses, where
 182 Components are needed. Component expressions are intended to be processed row-wise.

183

Operator	Category	Syntax	Description	Operand Data Sets	Component version
Round parenthesis ()	General purpose	Functional	Specifies the evaluation precedence	1	YES
assignment :=	General purpose	Infix	Assigns an Expression to a model artefact	2	NO
Membership #	General purpose	Infix	Identifies a Component within a Data Set	1	NO
get	General Purpose	Functional	Retrieves a Data Set	1..N	NO
put	General Purpose	Functional	Stores a Data Set	1	NO
eval	General Purpose	Functional	Evaluates an external routine	1	NO
length	String	Functional	Returns the length of a string	1	YES
concatenation +	String	Functional	Concatenates two strings	2	YES
trim	String	Functional	Eliminates trailing and leading whitespace from a String	1	YES
upper/lower	String	Functional	Makes a string upper / lower case	1	YES

substr	String	Functional	Extracts a substring from a string	1	YES
indexof	String	Functional	Returns the position of a String in another one	1	YES
unary plus +	Mathematical	Infix	Leaves the sign unaltered	1	YES
unary minus -	Mathematical	Infix	Changes the sign	1	YES
addition +, ++ and subtraction -, --	Mathematical	Infix	Sum or subtract	2	YES
multiplication *, ** and division /, //	Mathematical	Infix	Multiply or divide	2	YES
round	Mathematical	Functional	Rounds the values	1	YES
abs	Mathematical	Infix	Calculates the absolute value	1	YES
trunc	Mathematical	Infix	Truncates the values	1	YES
exp	Mathematical	Infix	Calculates the exponential	1	YES
ln	Mathematical	Infix	Calculates the natural logarithm	1	YES
log	Mathematical	Infix	Calculates the a base b logarithm	1	YES
power	Mathematical	Infix	Calculates the power	1	YES
nroot	Mathematical	Infix	Calculates the n-th root	1	YES
mod	Mathematical	Infix	Calculates the m modulo	1	YES
and	Boolean	Infix	Calculates the logical AND	2	YES
or	Boolean	Infix	Calculates the logical OR	2	YES
xor	Boolean	Infix	Calculates the logical XOR	2	YES
not	Boolean	Infix	Calculates the logical NOT	1	YES

union	Relational	Infix	Makes the set union	2	NO
intersect	Relational	Infix	Makes the set intersection	2	NO
symdiff	Relational	Functional	Makes the set symmetric difference	2	NO
setdiff	Relational	Infix	Makes the set difference	2	NO
merge	Relational	Infix	Carries out the join	2..N	NO
min/max	Statistical	Infix	Extracts the Data Point having minimum / maximum value	1	NO
hierarchy	Statistical	Infix	Aggregates following hierarchical links	1	NO
equal to =	Validation	Infix	Compares the values	2	YES
not equal to <>	Validation	Infix	Compares the values	2	YES
Greater than >, >=	Validation	Infix	Compares the values	2	YES
Less than <, <=	Validation	Infix	Compares the values	2	YES
in, not in	Validation	Infix	Verify if a value belongs to a set of values	2	YES
between	Validation	Infix	Verify if a value belongs to a set of values	2	YES
isnull	Validation	Functional	Compares the values with the NULL literal	1	YES
exists_in, not_exists_in, exists_in_all, not_exists_in_all	Validation	Functional	Checks the Identifiers and the foreign keys	2	NO
check	Validation	Functional	Checks if an expression verifies a condition	1	NO

match_characters	Validation	Functional	Checks for the presence of some characters	1	NO
match_values	Validation	Functional	Checks if the Value Domain Subset is respected	1	NO
if-then-else	Conditional	Functional	Makes different calculations according to a condition	1	YES
nvl	Conditional	Functional	Alters NULL values	1	YES
rename	General purpose	Clause (Postfix Operator)	Renames the Structure Components	1	YES (ONLY)
filter	Relational	Clause (Postfix Operator)	Filters the Data Points	1	YES (ONLY)
keep	Relational	Clause (Postfix Operator)	Alters the Data Structure	1	YES (ONLY)
calc	Relational	Clause (Postfix Operator)	Calculates the values of a Structure Component	1	YES (ONLY)
attrcalc	Relational	Clause (Postfix Operator)	Calculates the values of an Attribute	2..N	YES (ONLY)
aggregate	Statistical	Clause (Postfix Operator)	Calculates summary values for groups of Data Points	1	YES (ONLY)

185 Evaluation order of the Operators

186 Within a single expression, the operators are applied in sequence, according to the
 187 precedence order. Operators with the same precedence level are applied according to
 188 associativity rules. Precedence and associativity orders are reported in the following table.

189

Order	Operator	Description	Associativity
I	()	Round parenthesis. To alter the default order.	Left-to-right
II	get put eval length concatenation trim upper/lower substr indexof round abs trunc exp ln log power nRoot mod union intersect syndiff setdiff merge min/max hierarchy isnull check match_characters match_values nvl	The majority of the operators of the VTL	Left-to-right
III	# rename filter keep calc attrcalc aggregate	Membership & Clauses	Left-to-right

IV	unary plus unary minus not	Unary minus Unary plus Logical negation	Right-to-left
V	*, ** /, //	Multiplication Division	Left-to-right
VI	+, ++ -, --	Addition Subtraction	Left-to-right
VII	> >= < <= in, not in between	Greater than Less than In (not in) a value list In a range	Left-to-right
VIII	exists_in not_exists_in exists_in_all not_exists_in_all	Identifiers matching	Left-to-right
IX	= <>	Equal-to Not-equal-to	Left-to-right
X	and	Logical AND	Left-to-right
XI	or xor	Logical OR Logical XOR	Left-to-right
XII	if-then-else	Conditional (if-then-else)	Right-to-left
XIII	:=	Assignment	Right-to-left

191 Writing conventions for operators and types

192 In the remainder of the document, the following writing conventions (which are not part of
193 the language) are used when describing assignments, commands and operators.

194 For denoting the type of Parameter:

195 `ParameterTypeName<TypeName>`

196 For denoting Parameter types with more than one data type is:

197 `ParameterTypeName<TypeName1, TypeName2, ..., TypeNameN>`

198 Or, for brevity, with a regular expression formalism:

199 `ParameterTypeName<TypeName1+>` (one or more)

200 `ParameterTypeName<TypeName1*>` (zero or more)

201 When the data type is unspecified the following syntax is adopted:

202 `ParameterTypeName<?>`

203 To indicate a multiplicity of unspecified data types:

204 `ParameterTypeName<?+>` (one or more)

205 `ParameterTypeName<?*>` (zero or more)

206 When the data type in a Parameter type is unspecified, it can be templated (assigned a
207 name) in order to be cited elsewhere:

208 `ParameterTypeName<T>`

209 To indicate a multiplicity of templated data types:

210 `ParameterTypeName<T+>` (one or more)

211 `ParameterTypeName<T*>` (zero or more)

212 To assert that an artefact is of a certain data type is:

213 `typeName artefact`

214 Operator names and parameters are **case sensitive**.

215 In general, some operators have *infix style*, others have *functional style* and the clauses have
216 *postfix style*. Their syntax is described with **templates**, which are textual descriptions with the
217 following conventions:

218 `{optionalPart}`: an optional part is delimited by curly braces

219 `{one-or-more}+`: a syntactical part that is repeated from 1 to many occurrences

220 `{zero-or-more}*:` a syntactical part that is repeated from 0 to many occurrences

221 `[part1|part2|part3]`: alternative syntactical parts

222 `[part1|part2|part3]+`: alternative syntactical parts, from 1 to many occurrences

223 `[part1|part2|part3]*:` alternative syntactical parts, from 0 to many occurrences

224 Typically, in the description of operators there are type constraints that cannot be expressed
225 with this regular expression-like formalism. They are textually described in the “Constraints”
226 sections.

227 Example 1

```
228 Dataset<?+, MeasureComponent<Numeric>+> ds := abs  
229 ({dataset=}Dataset<?+, MeasureComponent<Numeric>+> ds_1)
```

230 The template above synthesizes:

- 231 • “:=”, **abs**, “(”, “)” and “dataset=” are reserved keywords;
- 232 • **abs** takes in input an expression `ds_1`, whose type is constrained to be a *Dataset* with
233 one or more Numeric *MeasureComponents*. “?” indicates that other Identifier or
234 *AttributeComponents* can be present (at least one *IdentifierComponent*).
- 235 • There can optionally be the “dataset=” keyword.
- 236 • It returns an expression `ds`, whose type is a *Dataset* with one or more Numeric
237 *MeasureComponents*. “?” indicates that other Identifier or *AttributeComponents* can
238 be present (at least one *IdentifierComponent*).

239

240 From this template, it is possible to infer some valid instances of the **abs** operator:

```
241 ds_1 := abs(ds_2)  
242 ds_1 := abs(dataset=ds_3)
```

243 Example 2

```
244 Dataset<?+, MeasureComponent<Numeric>+> ds := trunc  
245 ({dataset=}Dataset<?+, MeasureComponent<Numeric>+> ds_1,  
246 {decimals=}Constant<Integer> d)
```

247 The template above synthesizes:

- 248 • “:=”, **trunc**, “(”, “)”, “dataset=” and “decimals=” are reserved keywords
- 249 • **trunc** takes in input an expression `ds_1`, whose type is constrained to be a *Dataset*
250 with one or more Numeric *MeasureComponents*. “?” indicates that other Identifier or
251 *AttributeComponents* can be present (at least one *IdentifierComponent*).
- 252 • **trunc** takes in input an expression `d`, whose type is constrained to be a
253 *Constant<Integer>*
- 254 • There can optionally be the “dataset=” and “decimals=” keywords
- 255 • It returns an expression `ds`, whose type is a *Dataset* with one or more Numeric
256 *MeasureComponents*. “?” indicates that other Identifier or *AttributeComponents* can be
257 present (at least one *IdentifierComponent*).

258

259 From this template, it is possible to infer some valid instances of the **trunc** operator:

```
260 ds_1 := trunc(ds_2, 3)  
261 ds_1 := trunc(dataset=ds_3, 3)  
262 ds_1 := trunc(dataset=ds_3, decimals=3)  
263 ds_1 := trunc(ds_3, decimals=3)
```

264 Example 3

```
265 Dataset<?+, MeasureComponent<Numeric>+> ds := Dataset<?+,  
266 MeasureComponent<Numeric>+> ds_1 + Dataset<?+,  
267 MeasureComponent<Numeric>+> ds_2
```

268 The template above synthesizes:

- 269 • “:=”, “+” are reserved keywords of the operator
- 270 • “+” takes in input two expressions `ds_1` and `ds_2`, whose type is constrained to be a
271 *Dataset* with one or more *Numeric MeasureComponents* and at least one other
272 *(Identifier) Component*.
- 273 • It returns an expression `ds`, whose type is a *Dataset* with one or more *Numeric*
274 *MeasureComponents* and at least one other *(Identifier) Component*.

275 From this template, it is possible to infer some valid instances of the + operator, like the
276 following one:

```
277 ds_1 := ds_2 + ds_3
```

278 When no constraints on types need to be enforced, artefact names can be simply used. The
279 following assignment is an example:

```
280 artefact := expression
```


281 General purpose operators

282 round parentheses ()

283 Syntax

284 (expression)

285 Constraints

286 No specific constraints.

287 Semantics

288 The round parenthesis specify explicitly the order of evaluation of the operators

289 assignment :=

290 Syntax

291 variable parameter := expression

292 Constraints

293 No specific constraints.

294 Semantics

295 The assignment is the basic building block of VTL statements.

296 The “:=” symbol allows to assign the value of an expression to a variable parameter. An
297 expression may evaluate to any Composite or Collection data type (Datasets, Components,
298 Constants and Collections).

299 Examples

300 Assignment of a Constant<Numeric> value to a parameter:

301 numpi := 3.14

302 Assignment of a String value to a parameter:

303 str := "hello world"

304 Assignment of an expression to a parameter:

305 popA := populationDS + 1

306 Assignment of a Dataset expression to a parameter:

307 ds_1 := get("NAMESPACE/DF_NAME/2000.USD.M.F.A.BOP.ANN.STO.EABL")

308 Assignment of a Constant<Boolean> value to a parameter:

309 bool_var := true

310 membership #

311 Syntax

312 Dataset<?*,Component<?> c> ds
313 #
314 Component<?> c

315 Constraints

316 - c must be a Component of ds (static)

317 Semantics

318 The membership operator allows to select a single Component within the application of
319 another operator:

- 320 - c is a *Component*
321 - All the other *Components* are not considered in the application of the operator and are
322 preserved with unaltered roles. In case of duplications of names of Components, the
323 added ones will be suffixed with “_1”, though a renaming is advisable.

324 Parameters

- 325 - ds – the Dataset to be altered
326 - c – the Component that has been selected

327 Returns

328 Identifies a Component of the Dataset

329 Examples

330 The membership operator is particularly useful to work with operators that have specific
331 constraints in terms of the types of the *MeasureComponents* or have more than one
332 *MeasureComponent*.

333 Example 1

334 Suppose ds_1 is a multi-measure Dataset, where M1 is a *MeasureComponent*<Numeric> and
335 M2 is a *MeasureComponent*<String>; let ds_2 be a mono-measure Dataset with a single
336 *MeasureComponent*<Numeric> M1. ds_1 and ds_2 have the same *IdentifierComponents*. Let us
337 supposed the sum ds_1 + ds_2 is desired.

338 The Datasets cannot be directly summed, as ds_1 has a *MeasureComponent* that is not present
339 in ds_2 besides being non-numeric.

340 The following syntax: ds_1#M1 + ds_2 makes this assignment possible, as ds_1 is
341 temporarily considered mono-measure.

342 Example 2

343 Suppose the comparison operator (“=”) needs to be applied on the *IdentifierComponent*
344 COUNTRY of the Dataset ds_1. The comparison operator, as defined, acts on
345 *MeasureComponents*.

346 `ds_2 := ds_1#COUNTRY="Luxembourg"`

347 In this expression, the membership operator specifies that the *IdentifierComponent* COUNTRY
348 is temporarily considered as the only *MeasureComponent* to be used in the comparison.

349 Example 3

350 Suppose it is needed to round an *IdentifierComponent*. The round operator acts on
351 *MeasureComponents*, which must be all Numeric. Suppose we have a Dataset `ds_1` with a
352 *MeasureComponent*<String> DESCRIPTION and an *IdentifierComponent*<Numeric>
353 AVERAGE_AGE, which needs to be rounded to the 3rd decimal. The expression:

354 `ds_1 := round(ds_1#AVERAGE_AGE, 3)`

355 performs this task.

356 `ds_1#AVERAGE_AGE` temporarily considers `AVERAGE_AGE` the only
357 *MeasureComponent*<Numeric> of `ds_1`. The round is then normally applied and the
358 DESCRIPTION propagated.

359 Example 4

360 Let us suppose we have two multi-measure Datasets `ds_1` and `ds_2`, having the same
361 *IdentifierComponents* `K1` and `K2`, and the same *MeasureComponents* `M1` (which is a Numeric),
362 `M2` which is a String.

363 The expression

364 `ds_3 := ds_1#M1 + ds_2#M1`

365 sums only the *MeasureComponent* `M1`. The outcome Dataset has three *MeasureComponents*:
366 `M1`, which is obtained as the sum of `M1` in `ds_1` and in `ds_2`, `M2` originating from `ds_1` and
367 `M2_1` originating from `ds_2`.

368 **get**

369 Syntax

```
370 Dataset<?> ds_o := get
371     (PersistentDataset<?> ds_id {, PersistentDataset<?> ds_id}*
372     {, keep(keepPart Component<?> {, keepClause Component}*)}
373     {, filter(Component <Boolean> filterPart)}
374     {, aggregate(
375         [sum|avg|median|count|count_distinct|min|max] (
376         {include NULLS} MeasureComponent<Numeric> aggrPart)
377     {, [sum|avg|median|count|count_distinct|min|max] (
378         {include NULLS} MeasureComponent<Numeric> aggrPart)}*)
379     )}
380     )
```

381

382 Constraints

- 383 • *ds_id* is a PersistentDataset, whose name responds to the syntax explained in Section
384 “Data Identification Scheme” (static).
- 385 • All the input Datasets *ds_id* must have the same Logical Data Structure, which is the
386 same Components in number, name and type (static).
- 387 • *keepPart* must be a Component expression containing exactly the name of a
388 Component of any *ds* (complex Component expressions, combining more than one
389 Component are not allowed) (static).
- 390 • *aggrPart* must be a Component expression containing exactly the name of a
391 MeasureComponent present in any *ds* (no complex Component expressions, combining
392 more than one Component is allowed). If there is at least one *aggrPart*, there must be
393 one for each MeasureComponent that is present in a *keepPart*. If *keepPart* is omitted,
394 all MeasureComponents must be in the aggregate. This means that there cannot be
395 MeasureComponents, kept that are not used in aggregations (static).

396 Semantics

397 It is the data retrieval command. It allows to fetch all the instances of a PersistentDataset from
398 the system and returns a Dataset expression containing them. It takes in input a number (at
399 least one) of PersistentDatasets *ds*. Together with *put*, it is the only operator in VTL where a
400 PersistentDataset can be mentioned.

401 All the Datasets must have the same Logical Data Structure, which is the same Components in
402 number, name and type.

403 The command operates as follows: considers all the instances of the identified
404 PersistentDataset (selected according to the semantics of the identifier); builds a union with
405 duplicates (like SQL UNION ALL); keeps in the result only the Components that are present in
406 the *keepPart* (like SQL SELECT). If the keep part is omitted, all the Components are preserved
407 in the result; selects the only instances returning *true* for the *filterPart* Component<Boolean>
408 expression. For the *filterPart*, any complex Component <Boolean> expression over all the
409 Datasets Components (not only the ones mentioned in the *keepPart* can be used) and it is
410 evaluated row-wise (like SQL WHERE).

411 Finally, the command aggregates (like SQL aggregations and GROUP BY) applying an
412 aggregation function (**sum**: sum, **avg**: average, median: the statistical median, count: count of
413 occurrences, considering duplicates, **count_distinct**: count of occurrences, without
414 duplicates, **min**: minimum occurrence, **max**: maximum occurrence) over the
415 MeasureComponent specified in *aggrPart* grouping by the IdentifierComponents that are kept
416 in the *keepPart* (or all if there is no *keepPart*).

417

418 NULL values are considered in aggregations only if the *include NULLS* part is present.
419 Specifically, they propagate as usual resulting in a NULL sum, average or median if at least one
420 NULL is present among the values; in a NULL minimum or maximum if the only value to
421 aggregate coincides with NULL; they are considered as always distinct in both count and
422 *count_distinct*.

423 Viceversa, if *include NULLS* part is absent, NULL values are not considered in aggregations.

424 Parameters

425 PersistentDataset<?> *ds_id* – the PersistentDataset to be fetched

426 Component<?> *keepPart* - a Component name

427 Component<Boolean> *filterPart* – a Component<Boolean> expression which is evaluated row-
428 wise and states if a row is to be kept (if evaluates to *true*) or removed (if it does not evaluate
429 to true) from the result.

430 MeasureComponent<Numeric> *aggrPart* – the numeric MeasureComponent to aggregate.

431 Returns

432 DataSet<?> *ds_o* – a Dataset expression obtained as the union with duplicates of all the
433 Datasets specified by the identifiers *ds*, keeping only the columns specified in the *keepParts*
434 and the rows in the *filterParts*, aggregating over the all the MeasureComponents in *aggrParts*
435 grouping

436 Example 1

437 The expression:

```
438 ds_1 := get ("DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO", keep (K1,  
439 K2, M1))
```

440 Retrieves Dataset identified by

```
441 DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO
```

442 from the system, keeping the Identifier Components K1 and K2 and the Measure Component
443 M1.

444

DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO		
K1	K2	M1
1	A	5
2	B	7

445

ds_1		
K1	K2	M1
1	A	5
2	B	7

446

447 Example 2

448 The expression:

```

449     Ds_1 := get("DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO",
450               ("DF_NAME/2011-2012.USD.M.F.A.BOP.ANN.STO", keep(K1, K2, K3,
451               M1)))

```

452 Retrieves the union of datasets identified by

```

453     DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO
454     and
455     DF_NAME/2011-2012.USD.M.F.A.BOP.ANN.STO

```

456 keeping the Identifier Components K1, K2 and K3 and the Measure Component M1 for all of
457 them.

458

459

<i>DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO</i>			
<i>K1</i>	<i>K2</i>	<i>K3</i>	<i>M1</i>
1	A	X	5
2	B	Y	7

460

461

462

463

464

<i>DF_NAME/2011-2012.USD.M.F.A.BOP.ANN.STO</i>			
<i>K1</i>	<i>K2</i>	<i>K3</i>	<i>M1</i>
1	A	X	6
2	B	Y	7
3	C	Z	9

465

466

467

468

469

470

471

<i>ds_1</i>			
<i>K1</i>	<i>K2</i>	<i>K3</i>	<i>M1</i>
1	A	X	5
2	B	Y	7
2	B	Y	7
1	A	X	6
3	C	Z	9

472

473

474

475

476

477

478

479 Example 3

480 The expression:

```

481     ds_1 := get("DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO",
482               ("DF_NAME/2011-2012.USD.M.F.A.BOP.ANN.STO", keep(K1, K2, K3,
483               M1), filter(K3="X")))

```

484 retrieves the union of datasets identified by

485 `DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO`

486 and

487 `DF_NAME/2011-2012.USD.M.F.A.BOP.ANN.STO`

488 keeping the Identifier Components K1, K2 and K3 and the Measure Component M1 for all of
489 them and selecting only the rows where the value of the Component K3 equals to the
490 Constant<String> "X".

491

NAMESPACE/DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO			
K1	K2	K3	M1
1	A	X	5
2	B	Y	7

492

493

494

495

496

497

NAMESPACE/DF_NAME/2011-2012.USD.M.F.A.BOP.ANN.STO			
K1	K2	K3	M1
1	A	X	6
2	B	Y	7
3	C	Z	9

498

499

500

501

502

ds_1			
K1	K2	K3	M1
1	A	X	5
1	A	X	6

503

504 Example 4

505 The expression:

```
506 ds_1 := get(  
507 "DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO", "DF_NAME/2011-  
508 2012.USD.M.F.A.BOP.ANN.STO",  
509 keep(K1, K2, M1), aggregate(sum(M1)))
```

510 retrieves the union of datasets identified by

511 `DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO`

512 and `DF_NAME/2011-2012.USD.M.F.A.BOP.ANN.STO`

513 keeping the Identifier Components K1 and K2 and the Measure Component M1 for all of them.
 514 It aggregates over the MeasureComponent M1, grouping by the IdentifierComponents K1 and
 515 K2.

516

517

DF_NAME/2000- 2010.USD.M.F.A.BOP.ANN.STO				
K1	K2	K3	M1	M2
1	A	X	5	2
2	B	Y	7	3

518

519

520

521

522

523

DF_NAME/2011- 2012.USD.M.F.A.BOP.ANN.STO				
K1	K2	K3	M1	M2
1	A	Y	6	5
2	B	Y	7	7
3	C	Z	9	11

524

525

526

527

528

529

ds_1		
K1	K2	M1
1	A	11
2	B	14
3	C	9

530

531

532

533

534

535 Example 5

536 The expression:

537

```
ds_1 := get(
  "DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO", "DF_NAME/2011-
  2012.USD.M.F.A.BOP.ANN.STO",
  keep(K1, K2, M1,M2) filter(K3>5 or K3=1),
  aggregate(sum(M1),max(M2)))
```

538

539

540

541

542 retrieves the union of datasets identified by

543

```
DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO
and DF_NAME/2011-2012.USD.M.F.A.BOP.ANN.STO
```

544

545 keeping the Identifier Components K1 and K2 and the Measure Components M1 and M2 for all
 546 of them. It selects only the rows where K3 is greater than 5 or exactly 1. It aggregates over the
 547 MeasureComponent M1 by sum, over the MeasureComponent M2 by max, grouping by the
 548 IdentifierComponents K1 and K2.

549
550
551
552
553
554

DF_NAME/2000- 2010.USD.M.F.A.BOP.ANN.STO				
K1	K2	K3	M1	M2
1	A	10	5	2
2	B	1	7	3

DF_NAME/2011- 2012.USD.M.F.A.BOP.ANN.STO				
K1	K2	K3	M1	M2
1	A	25	6	5
2	B	1	7	7
3	C	3	9	11

555

ds_1			
K1	K2	M1	M2
1	A	11	5
2	B	14	7

556 **put**

557 Syntax

```
558 Dataset<?> ds_1 :=  
559 put  
560 (Dataset<?> ds, PersistentDataset<?> ds_id)
```

561 Constraints

- 562 • *ds_id* is a PersistentDataset, whose name responds to the syntax explained in Section
563 “Data Identification Scheme”(static).
- 564 • The Dataset *ds* must not contain duplicates, i.e. rows with the same values for all the
565 IdentifierComponents (static).
- 566 • The Logical Data Structure of *ds* must conform to the one of the Dataset in the system
567 that is identified by *ds_id* (static).

568 Semantics

569 Stores the content of a Dataset expression *ds* into the PersistentDataset *ds_id*. It returns a copy
570 of the input Dataset.

571 As long as a Dataset is only memorized in a parameter *ds*, it is not persistent. **Put** allows to
572 make it persistent, by mapping its content to a specific PersistentDataset in the system. As
573 explained in the specific section, *ds_id* can also select a particular portion of a Dataset. It

574 follows that in case *ds* has any Data Points whose Identifier Components are not conform to
575 the ones specified in *ds_id* or they are filtered out by filters possibly used in *ds_id*, they will be
576 discarded and not stored into the system.

577 Together with *get*, it is the only operator in VTL where a PersistentDataset can be mentioned.

578 Parameters

579 *ds* – the Dataset to be stored into the system.

580 *ds_id* – the PersistentDataset

581 Returns

582 *ds_1* – the output Dataset, which is a copy of the input one.

583 Example

```
584     ds := put(ds_1, " DF_NAME/2000-2010.USD.M.F.A.BOP.ANN.STO")
```

585 *ds_1* is stored.

```
586  
587     ds := put(log((ds_1 + ds_2), 10), "DF_NAME/2000-  
588     2010.USD.M.F.A.BOP.ANN.STO")
```

589 The result of logarithm is stored, while the sum is not persistent.

```
590  
591     ds := put(log(put(ds_1 + ds_2, "DF_NAME/2000-  
592     2011.USD.M.F.A.BOP.ANN.STO"), 10), " DF_NAME/2000-  
593     2010.USD.M.F.A.BOP.ANN.STO")
```

594 Both the results of the sum and the logarithm are stored into the system. The fact that put
595 outputs the input expression allows for this kind of use.

596 **eval**

597 Syntax

```
598     Dataset<?> ds_o :=  
599     eval  
600     (Constant<String> language,  
601     [{script=}Constant<String> script | Constant<String> programPath],  
602     {, {params=}ConstantList<?> parameterList}  
603     , {dataset=}PersistentDataset ds_id)
```

604 Constraints

- 605 • *language* must be the name of a programming language, meaningful and executable in
606 the target system (such as a SQL stored-procedure language, R, STATA, etc.) (dynamic).
- 607 • *script* must be the code of a program, valid with respect to the specified language. The
608 program can perform whatever internal logic, but is forced to calculate and
609 autonomously store exactly one PersistentDataset, *ds_id* (dynamic).

- 610 • *programPath* must be a valid path in the target system to a program file, compliant
611 with the specified language. It does not necessarily correspond to a filesystem file, but
612 can also be the identifier of a DBMS stored procedure, and so forth (dynamic).
613 • *parameterList* must be compatible in order and type with the input parameters of the
614 script (dynamic).

615 Semantics

616 Eval operator allows executing a user-defined program to calculate a Persistent Dataset.

617 The program is user-defined and can perform any internal logic, however it has to adhere to
618 some conventions:

- 619 - it can take as input only String or Numeric parameters, which are directly bound to
620 *parameterList*;
621 - it must autonomously store its results into a single PersistentDataset *ds_id*. Indeed, the
622 operator fetches the saved Dataset (like a common **get** operation) and returns it as output,
623 which can be handled within other VTL expressions;
624 - it must calculate exactly one Dataset;
625 - it cannot refer to a parameter variable, but can only work with physical objects, such as
626 relational tables (for SQL), data frames (for R), which are loaded autonomously by the
627 program with the appropriate commands. Therefore, if a Dataset that has been calculated
628 in a previous step needs to be used within a user-defined program, it must be stored (with
629 a **put**) into the system and loaded appropriately by the program logic afterwards;
630 - it must return 0 if it has terminated correctly, a negative number otherwise.

631

632 Parameters

633 Constant<String> *language* – the programming language of the script

634 Constant<String> *script* - the code of the script

635 Constant<String> *programPath* – a path to a script file

636 ConstantList<?> *parameterList* – the List of input parameters for the script

637 PersistentDataset<?> *ds_id* – the PersistentDataset the program saves into

638

639 Returns

640 DataSet<?> *ds_o* – the Dataset whose content has been calculated by the user-defined script.

641 Example

642 The following example shows the usage of a user-defined function that calculates the n-th
643 power of a Dataset. Indeed, this would be meaningless as VTL already provides a native
644 operator for such operation, however it shows the potentialities of **eval**.

```
645     ds_o := eval("PL/SQL", "programs.dsToN", [5],  
646     "NAMESPACE/DF_NAME/2010.USD.M.F.A.BOP.ANN.STO")
```

647 where `programs.dsToN` contains the following code:

```
648     create function programs.dsToN(n number)
```

```
649     return number is
650
651     begin
652     insert into 2010.USD.M.F.A.BOP.ANN.STO
653         select dim1, dim2, ..., power(measure,n)
654         from 2010.USD.M.F.A.BOP.ANN.STO2;
655     except
656         when others then
657         return -1;
658     return 0;
659     end;
```

661 The reported stored function takes in input a number n and elevates every Data Point within
662 PersistentDataset 2010.USD.M.F.A.BOP.ANN.STO2 to the n-th power and stores the result
663 into PersistentDataset 2010.USD.M.F.A.BOP.ANN.STO.

664 It returns 0 if the operation succeeded, -1 otherwise.

665 At the end of a correct execution, *ds_o* will contain the fetched content of the
666 PersistentDataset 2010.USD.M.F.A.BOP.ANN.STO.

667 String operators

668 length

669 Component version

670 Syntax

```
671     Component<Integer> c :=  
672     length  
673     ({component=}Component<String> c_1)
```

674 Constraints

675 None

676 Semantics

677 Returns the length of *c_1*.

678 Example

679 If A = "Hello, World!":

```
680 B := length(A) // B = 13
```

681 Dataset version

682 Syntax

```
683     Dataset<?+, MeasureComponent<Integer>+> ds_2 :=  
684     length  
685     ({dataset=}Dataset<?+, MeasureComponent<String>+> ds_1)
```

686 Constraints

687 - *ds_1* only has MeasureComponents<String> (static).

688 Semantics

689 Get length of a string. It takes in input a Dataset *ds_1* and returns another Dataset *ds_2* with
690 the same IdentifierComponents, AttributeComponents (according to attribute propagation
691 rules) and a MeasureComponent<Integer> for each MeasureComponent<String> in *ds_1* with
692 the same name. For each row in *ds_1* there is a row in *ds_2* where the MeasureComponents are
693 obtained by calculating the length of the String in the original MeasureComponent.

694 Parameters

695 *ds_1* – the input Dataset

696 Returns

697 *ds_2* – the output Dataset with the lengths

698 Example

699 `ds_r := length(ds_1)`

700

ds_1		
K1	K2	M1
1	A	"hello"
2	B	""

701

ds_r		
K1	K2	M1
1	A	5
2	B	0

702

703 **concatenation** +

704 **Component version**

705 Syntax

```
706     Component<String> c :=  
707     Component<String> c_1  
708     +  
709     Component<String> c_2
```

710 Constraints

711 None

712 Semantics

713 Returns the concatenation of *c_1* and *c_2*.

714 Example

715 If A = "Hello,", B = " world!":

716 `C := A + B // C = "Hello, world!"`

717 Dataset version

718 Syntax

```
719 Dataset<?+, MeasureComponent<String>+> ds_3 :=  
720 [Dataset<?+, MeasureComponent<String>+>|Constant<String>] ds_1  
721 +  
722 [Dataset<?+, MeasureComponent<String>+>|Constant<String>] ds_2
```

723 Constraints

- 724 • At least one between *ds_1* and *ds_2* must be a Dataset (static).
- 725 • If both *ds_1* and *ds_2* are Datasets, they must have the same IdentifierComponents<?>,
726 which must be equal in name and type or, at least, the IdentifierComponents of one
727 must be a subset of the IdentifierComponents of the other (static).
- 728 • If both *ds_1* and *ds_2* are Datasets, they must have the same MeasureComponents
729 (static).
- 730 • If *ds_1* is a Dataset, it must have only String MeasureComponents (static).
- 731 • If *ds_2* is a Dataset, it must have only String MeasureComponents (static).

732 Semantics

733 Takes in input two Datasets *ds_1* and *ds_2*, with at least one String MeasureComponent, and
734 such that the IdentifierComponents of *ds_1* are a subset of the IdentifierComponents of *ds_2* or
735 vice-versa and *ds_1* and *ds_2* have the same MeasureComponents, which equal in names and
736 types.

737 Alternatively, takes in input one Dataset and a Constant<String>, without other constraints.

738 Returns a third Dataset *ds_3* with the all the IdentifierComponents that are in *ds_1* or in *ds_2*
739 (without duplicates) all the MeasureComponents (without duplicates) and
740 AttributeComponents (according to the attribute propagation rules) containing for each pair
741 of rows in *ds_1* and *ds_2* with the same values for all the common IdentifierComponents, a row
742 with those values for the common IdentifierComponents and the ones of the most
743 comprehensive Dataset for the other IdentifierComponents and the concatenation of the
744 values in the corresponding MeasureComponents.

745 If *ds_2* (or *ds_1*) is a Constant<String>, the operator returns a second Dataset *ds_3* with all the
746 IdentifierComponents in *ds_1* (*ds_2*) with the same values, all the MeasureComponents in *ds_1*
747 (*ds_2*), whose values are the concatenation with the value of the Constant<String> *ds_2* (*ds_1*).

748 Parameters

749 *ds_1* - a Dataset with at least one MeasureComponent<String> or a Constant<String>.

750 *ds_2* - a Dataset with at least one MeasureComponent<String> or a Constant<String>.

751 Returns

752 *ds_3* - a Dataset that is the concatenation two according to the described semantics.

753 Example

```
754 ds_r := ds_1 + " " + ds_2
```

755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771

ds_1		
K1	K2	M1
1	A	"hello"
2	B	"hi"

ds_2		
K1	K2	M1
1	A	"world"
2	B	"there"

ds_r		
K1	K2	M1
1	A	"hello world"
2	B	"hi there"

772 **trim**

773 **Component version**

774 Syntax

```
775     Component<String> c :=  
776     trim  
777     ({component=}Component<String> c_1)
```

778 Constraints

779 None

780 Semantics

781 Trims whitespace from left and right of string.

782 Example

783 If A = " Hello, world! ":

784 B := trim(A) // B = "Hello, world!"

785 **Dataset version**

786 Syntax

```
787 Dataset<?+, MeasureComponent<String>+> ds_2  
788 trim  
789 ({dataset=}Dataset<?+, MeasureComponent<String>+> ds_1)
```

790 Constrains

791 - *ds_1* only has MeasureComponents<String> (static).

792 Semantics

793 It takes in input a Dataset *ds_1* and returns another Dataset *ds_2* with the same
794 IdentifierComponents , AttributeComponents (according to attribute propagation rules) and a
795 MeasureComponent<String> for each MeasureComponent<String> in *ds_1* with the same
796 name. For each row in *ds_1* there is a row in *ds_2* where the MeasureComponents are obtained
797 by eliminating leading and trailing whitespace from the String in the original
798 MeasureComponent.

799 Parameters

800 *ds_1* - the input Dataset.

801 Returns

802 *ds_2* - the output Dataset

803 Example

```
804 r := trim(ds_1)
```

C1		
K1	K2	M1
1	A	"hello world"
2	B	"hi"
3	C	"help!"

805

R		
K1	K2	M1
1	A	"hello world"
2	B	"hi"
3	C	"help!"

806

807 upper/lower

808 Component version

809 Syntax

```
810     Component<String> c :=  
811     [upper|lower]  
812     ({component=}Component<String> c_1)
```

813 Constraints

814 None

815 Semantics

816 Makes a string uppercase/ lowercase.

817 Example

818 If A = "Hello, World!":

```
819 B := upper(A) // B = "HELLO, WORLD!"
```

```
820 B := lower(A) // B = "hello, world!"
```

821 Dataset version

822 Syntax

```
823     Dataset<?+, MeasureComponent<String>+> ds_2 :=  
824     [upper|lower]  
825     ({dataset=}Dataset<?+, MeasureComponent<String>+> ds_1)
```

826 Constraints

827 - *ds_1* only has MeasureComponents<String> (static).

828 Semantics

829 It takes in input a Dataset *ds_1* and returns another Dataset *ds_2* with the same
830 IdentifierComponents, AttributeComponents (according to attribute propagation rules) and a
831 MeasureComponent<String> for each MeasureComponent<String> in *ds_1* with the same
832 name. For each row in *ds_1* there is a row in *ds_2* where the MeasureComponents are obtained
833 by making uppercase (lowercase) the String in the original MeasureComponent. Non-
834 alphabetic characters are simply copied.

835 Parameters

836 *ds_1* - the input Dataset.

837 Returns

838 *ds_2* - the output Dataset

839 Example

840 `ds_r := upper(ds_1)`

ds_1		
K1	K2	M1
1	A	“hello world”
2	B	“hi”
3	C	“help”

841

ds_r		
K1	K2	M1
1	A	“HELLO WORLD”
2	B	“HI”
3	C	“HELP”

842

843 **substr**

844 **Component version**

845 Syntax

```
846 Component<String> c :=  
847 substr  
848 ({component=}Component<String> c_1,  
849 {startPosition=}Constant<Integer> startPosition {,  
850 {length=}Constant<Integer> length})
```

851 Constraints

852 $0 \leq \text{startPosition} < \text{length}(c_1); \text{startPosition} + \text{length} \leq \text{length}(c_1)$ (dynamic)

853 Notice that if $\text{startPosition} + \text{length} > \text{length}(c_1)$, then length is ignored.

854 Semantics

855 Returns the substring of `c_1` starting from position `startPosition` and extracting `length`
856 characters. 0 is the initial position of the Strings. If `length` is omitted, all the remaining
857 characters in the String are considered.

858

859 Examples

```
860     Assuming that A = "Hello, world!":
861     B := substr(A, 2) // B = "llo, world!"
862     B := substr(A, 2, 5) // B = "llo, "
863     B := substr(A, 0, 4) // B = "Hell"
```

864 **Dataset version**

865 Syntax

```
866     Dataset<?+, MeasureComponent<String>+> ds_2 :=
867     substr(
868     {dataset=}Dataset<?+, MeasureComponent<String>+> ds_1,
869     {startPosition=}Constant<Integer> startPosition,
870     {, {length=} Constant<Integer> length }
871     )
```

872 Constraints

- 873 - *ds_1* only has MeasureComponents<String> (static).
- 874 - $0 \leq \text{startPosition} < \text{length}(c_1)$; $\text{startPosition} + \text{length} \leq \text{length}(c_1)$ (dynamic)
- 875 Notice that if $\text{startPosition} + \text{length} > \text{length}(c_1)$, then length is ignored.

876 Semantics

877 It takes in input a Dataset *ds_1* and returns another Dataset *ds_2* with the same
878 IdentifierComponents, AttributeComponents (according to attribute propagation rules) and a
879 MeasureComponent<String> for each MeasureComponent<String> in *ds_1* with the same
880 name.

881 For each row in *ds_1* there is a row in *ds_2* where the Strings in the MeasureComponents are
882 obtained from the original ones in *ds_1* by taking the substring starting from position
883 *startPosition* and extracting *length* characters. 0 is the initial position. If length is omitted, all
884 the remaining characters in the String are considered. If *startPosition* exceeds the String
885 length, NULL is returned. If *length* exceeds the String length, all the characters of the String
886 from *startPosition* are returned.

887 Parameters

- 888 - *ds_1* – the input Dataset
- 889 - *startPosition* – the position to start extracting from. 0 is the beginning of the String
- 890 - *length* – the number of characters to extract from *startPosition*

891 Returns

892 *ds_2* - the output Dataset

893

894 Example 1

```
895     ds_r := substr(ds_1,7)
```

ds_1		
K1	K2	M1
1	A	"hello world"

896

ds_r		
K1	K2	M1
1	A	"orld"

897

898 Example 2

899 `ds_r := substr(ds_1,0,5)`

ds_1		
K1	K2	M1
1	A	"hello world"

900

ds_r		
K1	K2	M1
1	A	"hello"

901

902 **indexof**

903 **Component version**

904 Syntax

```
905 Component<Numeric> c :=
906 indexof
907 ({component=}Component<String> c_1, {string=}Constant<String>
908 stringToSearch)
```

909 Constraints

910 None

911 Semantics

912 It takes in input a *Component*<String> *c_1* and a *Constant*<String> *stringToSearch* and
913 returns a *Component*<Numeric>, whose value is the starting position of *stringToSearch* in
914 *c_1* (starting from 0).

915 If *stringToSearch* is not found, -1 is returned. If *stringToSearch* is empty or NULL, -1 is
916 returned.

917 Example

918 If A = "Hello, world!":
919 B := indexof(A, "Hello") // B = 0
920 B := indexof(A, "world") // B = 7
921 B := indexof(A, "cat") // B = -1

922 **Dataset version**

923 Syntax

```
924 Dataset<?+, MeasureComponent<Numeric>+> ds_2 :=  
925 indexof  
926 ({dataset=}Dataset<?+, MeasureComponent<String>+> ds_1,  
927 {string=}Constant<String> stringToSearch)
```

928 Constraints

929 - *ds_1* only has *MeasureComponents*<String> (static).

930 Semantics

931 It takes in input a *Dataset* *ds_1* and a *Constant*<String> *stringToSearch* and returns
932 another *Dataset* *ds_2* with the same *IdentifierComponents*, *AttributeComponents* (according to
933 attribute propagation rules) and a *MeasureComponent*<Numeric> for each
934 *MeasureComponent*<String> in *ds_1* with the same name.

935 For each row in *ds_1* there is a row in *ds_2* where the numeric values in the
936 *MeasureComponents* are the starting position of *stringToSearch* in the respective
937 *MeasureComponent*<String> in *ds_1* (starting from 0).

938 If *stringToSearch* is not found, -1 is returned. If *stringToSearch* is empty or NULL, -1 is
939 returned.

940 Parameters

941 - *ds_1* - the input *Dataset*.
942 - *stringToSearch* - the *Constant*<String> to search.

943 Returns

944 *ds_2* - the output *Dataset*

945 Example

946 `ds_2 := indexof(ds_1, "hello")`

ds_1		
K1	K2	M1
1	A	"hello world"
2	A	"say hello"
3	A	"he"
4	A	"hi, hello!"

947

ds_2		
K1	K2	M1
1	A	0
2	A	4
3	A	-1
4	A	4

948

949 Mathematical operators

950 unary plus +

951 **Component version**

952 Syntax

```
953     Component<Numeric> c :=  
954     +  
955     Component<Numeric> c_1
```

956 Constraints

957 None

958 Semantics

959 No effect. Returns *c_1* without altering its sign.

960 Example

```
961 A := +B      //if B = 5, then A = 5
```

962 **Dataset version**

963 Syntax

```
964     ds_o :=  
965     +  
966     Dataset<?+, MeasureComponent<Numeric>+> ds
```

967 Constraints

968 None

969 Semantics

970 Returns the Dataset *ds* without changing any sign.

971 Parameters

972 *ds* – a Dataset with at least a Numeric MeasureComponent.

973 Returns

974 *ds_o* – A copy of *ds* with no alterations.

975 Example

```
976     ds_2 := +ds_1
```

977

ds_1		
K1	K2	M1
1	A	11
2	B	14
3	C	9

978

ds_2		
K1	K2	M1
1	A	11
2	B	14
3	C	9

979

980 **unary minus** -

981 **Component version**

982 Syntax

983 Component<Numeric> c :=

984 -

985 Component<Numeric> c_1

986 Constraints

987 None

988 Semantics

989 Returns c changing its sign.

990 Examples

991 A := -B //if B = 5, then A = -5

992 C := -A //A = 5

993 **Dataset version**

994 Syntax

995 ds_o :=

996 -

997 Dataset<?+, MeasureComponent<Numeric>+> ds

998 Constraints

999 None

1000 Semantics

1001 Returns the Dataset *ds* where the signs of all the MeasureComponent<Numeric> are inverted.
1002 Differently typed MeasureComponents (MeasureComponent<String>,
1003 MeasureComponent<Boolean>) are ignored and kept unchanged in the result.

1004 Parameters

1005 *ds* – a Dataset expression with at least a Numeric MeasureComponent.

1006 Returns

1007 *ds_o* – A copy of *ds* where the signs of all the numeric Measure Components are inverted.

1008 Example 1

1009 `ds_2 := -ds_1`

1010

ds_1		
K1	K2	M1
1	A	11
2	B	14
3	C	9

1011

ds_2		
K1	K2	M1
1	A	-11
2	B	-14
3	C	-9

1012 Example 2

1013 `ds_2 := -ds_1`

ds_1				
K1	K2	M1	M2	M3
1	A	11	12	“A”
2	B	14	13	“B”
3	C	9	14	“C”

1014

ds_2				
K1	K2	M1	M2	M3
1	A	-11	-12	"A"
2	B	-14	-13	"B"
3	C	-9	-14	"C"

1015

1016 addition and subtraction + ++ - --

1017 Component version

1018 Syntax

```

1019 Component<Numeric> c := Component<Numeric> c_1
1020 [+|-]
1021 Component<Numeric> c_2

```

1022 Constraints

1023 None

1024 Semantics

1025 Calculates the algebraic sum, of c_1 and c_2, or c_1 and -c_2.

1026 Example

1027 For A = 5 and B = 4:

```

1028 C := A + B // C = 9
1029 C := A - B // C = 1

```

1030 Dataset version (implicit and explicit zero)

1031 Syntax

1032 Explicit zero:

```

1033 Dataset<?+, MeasureComponent<Numeric>+> ds_3 :=
1034 [Dataset<?+, MeasureComponent<Numeric>+>|Constant<Numeric>] ds_1
1035 [++|--]
1036 [Dataset<?+, MeasureComponent<Numeric>+>|Constant<Numeric>] ds_2

```

1037 Implicit zero:

```

1038 Dataset<?+, MeasureComponent<Numeric>+> ds_3 :=
1039 [Dataset<?+, MeasureComponent<Numeric>+>|Constant<Numeric>] ds_1
1040 [+|-]

```

1041 [Dataset<?+, MeasureComponent<Numeric>+>|Constant<Numeric>] ds_2

1042 Constraints

- 1043 • At least one between *ds_1* and *ds_2* must be a Dataset (static).
- 1044 • If both *ds_1* and *ds_2* are Datasets, they must have the same IdentifierComponents<?>, which must be equal in name and type or, at least, the IdentifierComponents of one must be a subset of the IdentifierComponents of the other (static).
- 1047 • If both *ds_1* and *ds_2* are Datasets, they must have the same MeasureComponents (static).
- 1048 • If *ds_1* is a Dataset, it must have only numeric MeasureComponents (static).
- 1049 • If *ds_2* is a Dataset, it must have only numeric MeasureComponents (static).

1051 Semantics

1052 Takes in input two Datasets *ds_1* and *ds_2*, with at least one Numeric MeasureComponent, and
1053 such that the IdentifierComponents of *ds_1* are a subset of the IdentifierComponents of *ds_2* or
1054 vice-versa and *ds_1* and *ds_2* have the same MeasureComponents, which equal in names and
1055 types.

1056 Alternatively, takes in input one Dataset and a Constant<Numeric>, without other constraints.

1057 Explicit zero

1058 [++|--]

1059 Returns a third Dataset *ds_3* with the all the IdentifierComponents that are in *ds_1* or in *ds_2*
1060 (without duplicates) all the MeasureComponents (without duplicates) and
1061 AttributeComponents (according to the attribute propagation rules) containing for each pair
1062 of rows in *ds_1* and *ds_2* with the same values for all the common IdentifierComponents, a row
1063 with those values for the common IdentifierComponents and the ones of the most
1064 comprehensive Dataset for the other IdentifierComponents, the algebraic sum (difference) of
1065 the values in the corresponding MeasureComponents.

1066

1067 *Explicit zero* → *implicit missing*

1068

1069 As this version of the operator returns a row in *ds_3* only for each pair of matching rows in
1070 *ds_1* and *ds_2*, rows that are present only in one between *ds_1* and *ds_2* are assumed to be
1071 implicitly missing in the other (since the zero is represented explicitly) and then, not
1072 produced in the output.

1073 Implicit zero

1074 [+|-]

1075 Returns a third Dataset *ds_3* with the all the common IdentifierComponents that are in *ds_1*
1076 or in *ds_2*, all the MeasureComponents (one for each homonym pair and with that name and
1077 type) and AttributeComponents (according to the attribute propagation rules) containing for
1078 each row in *ds_1* and *ds_2* a row with those values for the common IdentifierComponents and
1079 the ones of the most comprehensive Dataset for the other IdentifierComponents and the
1080 algebraic sum (difference) of the values in the corresponding MeasureComponents. Moreover
1081 *ds_3* contains all the rows in *ds_1* without corresponding ones in *ds_2* and viceversa.
1082

1083 *Implicit zero* → *Explicit missing*

1084

1085 As this version of the operator returns a row in *ds_3* for each row in *ds_1* or *ds_2*,
1086 independently of any matching of the IdentifierComponents, rows that are present only in one
1087 between *ds_1* and *ds_2* are assumed to be present in the other with value 0 for all the
1088 MeasureComponents. Therefore, missing values should be explicitly represented.

1089

1090 In both explicit and implicit cases, if *ds_2* (or *ds_1*) is a Constant<Numeric>, the operator
1091 returns a second Dataset *ds_3* with all the IdentifierComponents in *ds_1* (*ds_2*) with the same
1092 values, all the MeasureComponents in *ds_1* (*ds_2*), whose values are the algebraic sum
1093 (difference) with the value of the Constant<Numeric> *ds_2* (*ds_1*).

1094 Parameters

1095 *ds_1* – a Dataset with at least one MeasureComponent<Numeric> or a Constant<Numeric>

1096 *ds_2* – a Dataset with at least one MeasureComponent<Numeric> or a Constant<Numeric>

1097 Returns

1098 *ds_3* – a Dataset that is the algebraic sum of the two according to the described semantics.

1099

1100 Example

1101 In this example, we calculate the total population of a set of countries given two Datasets: one
1102 of the male population, and another, of the female population. They contain one measure
1103 each. Thus, the result will contain a single measure with the results of the addition.

1104

ds_1		
TIME	GEO	POPULATION
2013	Belgium	5
2013	Denmark	2
2013	France	3
2013	Spain	4

1105

ds_2			
TIME	GEO	AGE	POPULATION
2013	Belgium	Total	10
2013	Greece	Total	11
2013	Belgium	Y15-24	3
2013	Greece	Y15-24	2
2013	Spain	Y15-24	6

1106 The result of the expression `ds_3 := ds_1 + ds_2` (an implicit zero addition) will be:

1107

ds_3			
TIME	GEO	AGE	POPULATION
2013	Belgium	Total	15
2013	Belgium	Y15-24	8
2013	Denmark	NULL	2
2013	France	NULL	3
2013	Greece	Total	11
2013	Greece	Y15-24	2
2013	Spain	Y15-24	10

1108

1109 The result of the expression `ds_3 := ds_1 ++ ds_2` (explicit zero addition) will be:

1110

ds_3			
TIME	GEO	AGE	POPULATION
2013	Belgium	Total	15
2013	Belgium	Y15-24	8
2013	Spain	Y15-24	10

1111

1112

1113 **Multiplication and division** * ** / //

1114 **Component version**

1115 Syntax

1116 Component<Numeric> c := Component<Numeric> c_1

1117 [*|/]

1118 Component<Numeric> c_2

1119 Constraints

1120 For division (/) `c_2 <> 0` (dynamic).

1121 Semantics

1122 Calculates the product of c_1 and c_2 .

1123 Calculates the ratio between c_1 and c_2 . The case of $c_2 = 0$, should result into a runtime
1124 exception .

1125 Example

1126 For $A = 5$ and $B = 4$:

1127 $C := A * B // C = 20$

1128 $C := A / B // C = 1.25$

1129 **Dataset version (implicit zero and explicit zero)**

1130 Syntax

1131 **Explicit zero:**

```
1132 Dataset<?+, MeasureComponent<Numeric>+> ds_3 :=  
1133 [Dataset<?+, MeasureComponent<Numeric>+>|Constant<Numeric>] ds_1  
1134 [**|//]  
1135 [Dataset<?+, MeasureComponent<Numeric>+>|Constant<Numeric>] ds_2
```

1136 **Implicit zero:**

```
1137 Dataset<?+, MeasureComponent<Numeric>+> ds_3 :=  
1138 [Dataset<?+, MeasureComponent<Numeric>+>|Constant<Numeric>] ds_1  
1139 [*|//]  
1140 [Dataset<?+, MeasureComponent<Numeric>+>|Constant<Numeric>] ds_2
```

1141 Constraints

- 1142 • At least one between ds_1 and ds_2 must be a Dataset (static).
- 1143 • If both ds_1 and ds_2 are Datasets, they must have the same IdentifierComponents<?>, which must be equal in name and type or, at least, the IdentifierComponents of one must be a subset of the IdentifierComponents of the other (static).
- 1144 • If both ds_1 and ds_2 are Datasets, they must have the same MeasureComponents (static).
- 1145 • If ds_1 is a Dataset ,it must have only numeric MeasureComponents (static).
- 1146 • If ds_2 is a Dataset, it must have only numeric MeasureComponents (static).
- 1147 • If ds_2 is a Constant<Numeric>, it cannot be zero (dynamic).

1151 Semantics

1152 Takes in input two Datasets ds_1 and ds_2 , with at least one Numeric MeasureComponent, and
1153 such that the IdentifierComponents of ds_1 are a subset of the IdentifierComponents of ds_2 or
1154 viceversa and ds_1 and ds_2 have the same MeasureComponents, which equal in names and
1155 types.

1156 Alternatively, takes in input one Dataset and a Constant<Numeric>, without other constraints.

1157

1158 Explicit zero
1159 **[**|//]**
1160 Returns a third Dataset ds_3 with the all the IdentifierComponents that are in ds_1 or in ds_2
1161 (without duplicates) all the MeasureComponents (without duplicates) and
1162 AttributeComponents (according to the attribute propagation rules) containing for each pair
1163 of rows in ds_1 and ds_2 with the same values for all the common IdentifierComponents, a row
1164 with those values for the common IdentifierComponents and the ones of the most
1165 comprehensive Dataset for the other IdentifierComponents and the product (division) of the
1166 values in the corresponding MeasureComponents.

1167
1168 *Explicit zero* → *implicit missing*
1169

1170 As this version of the operator returns a row in ds_3 only for each pair of matching rows in
1171 ds_1 and ds_2 , rows that are present only in one between ds_1 and ds_2 are assumed to be
1172 implicitly missing in the other (since the zero is represented explicitly) and then, not
1173 produced in the output.
1174

1175 Implicit zero

1176 **[*|/]**

1177 Returns a third Dataset ds_3 with the all the common IdentifierComponents that are in ds_1
1178 or in ds_2 , all the MeasureComponents (one for each homonym pair and with that name and
1179 type) and AttributeComponents (according to the attribute propagation rules) containing for
1180 each row in ds_1 and ds_2 a row with those values for the common IdentifierComponents and
1181 the ones of the most comprehensive Dataset for the other IdentifierComponents and the
1182 product (division) of the values in the corresponding MeasureComponents. Moreover ds_3
1183 contains all the rows in ds_1 without corresponding ones in ds_2 and viceversa.

1184
1185 *Implicit zero* → *Explicit missing*
1186

1187 As this version of the operator returns a row in ds_3 for each row in ds_1 or ds_2 ,
1188 independently of any matching of the IdentifierComponents, rows that are present only in one
1189 between ds_1 and ds_2 are assumed to be present in the other with value 0 for all the
1190 MeasureComponents. Therefore, missing values should be explicitly represented.

1191
1192 In both explicit and implicit cases, if ds_2 (or ds_1) is a Constant<Numeric>, the operator
1193 returns a second Dataset ds_3 with all the IdentifierComponents in ds_1 (ds_2) with the same
1194 values, all the MeasureComponents in ds_1 (ds_2), whose values are the product (division)
1195 involving the value of the Constant<Numeric> ds_2 (ds_1) as numerator or denominator.

1196 Parameters

1197 ds_1 – a Dataset with at least one MeasureComponent<Numeric> or a Constant<Numeric>.

1198 ds_2 – a Dataset with at least one MeasureComponent<Numeric> or a Constant<Numeric>.

1199 Returns

1200 ds_3 – a Dataset that is the sum (division) of the two according to the described semantics.

1201 Example

total_population				
TIME	GEO	AGE	POPULATION	UNEMPLOYMENT_RATE
2012	Belgium	Total	100	7.6
2012	Greece	Total	10	24.3
2012	Spain	Total	20	25
2012	Belgium	Y15-24	30	3.6
2012	Greece	Y15-24	5	18.3
2012	Switzerland	Y15-24	2	20

1202

Overcrowding_rate_urbanization		
TIME	GEO	PERCENTAGE
2012	Belgium	0.01
2012	Greece	0.1
2012	Spain	0.2
2012	Malta	0.3
2012	Finland	0.4
2012	France	0.5

1203 The result of the expression

1204 `total_population[rename POPULATION as PERCENTAGE]#PERCENTAGE *`

1205 `Overcrowding_rate_urbanization#PERCENTAGE`

1206 is the following dataset:

TIME	GEO	AGE	PERCENTAGE
2012	Belgium	Total	1
2012	Greece	Total	1
2012	Spain	Total	4
2012	Belgium	Y15-24	0.3
2012	Greece	Y15-24	0.5

1207

1208 round

1209 Component version

1210 Syntax

```
1211     Component<Numeric> c :=  
1212     round  
1213     ({component=}Component<Numeric> c_1, {decimals=}Constant<Integer>  
1214     decimals)
```

1215 Constraints

1216 None

1217 Semantics

1218 Rounds *c_1* to the number of decimal digits specified by *decimals*.

1219 Examples

1220 Let us assume $P = 3.14159$

```
1221     A := round(P, 2) // A = 3.14
```

```
1222     B := round(P, 4) // B = 3.1416
```

1223 Dataset version

1224 Syntax

```
1225     Dataset<?+, MeasureComponent<Numeric>+> ds_2 :=  
1226     round  
1227     ({dataset=}Dataset<?+, MeasureComponent<Numeric>+> ds_1,  
1228     {decimals=}Constant<Integer> decimals)
```

1229 Constraints

- 1230 - *ds_1* only has Numeric MeasureComponents (static).
- 1231 - *decimals* > 0 (static).

1232 Semantics

1233 It takes in input a Dataset *ds_1* and returns another Dataset *ds_2* with the same
1234 IdentifierComponents, AttributeComponents (according to attribute propagation rules) and
1235 the same MeasureComponents. For each row in *ds_1* there is a row in *ds_2* where the
1236 MeasureComponents are rounded to *decimals* decimal positions.

1237 Parameters

1238 *ds_1* – the Dataset to round

1239 *decimals* – the decimal positions to round to

1240 Returns

1241 *ds_2* – the rounded Dataset

1242 Example

1243 Let us start from the the dataset:

unemployment					
AGE	TIME	GEO	SEX	YOUTH_UNEMPLOYMENT	UNEMPLOYMENT
From 20 to 29 years	2011	Germany	Total	7.5	5.9
From 20 to 29 years	2012	Germany	Total	7.1	5.5
From 20 to 29 years	2011	Greece	Total	33.7	17.7
From 20 to 29 years	2012	Greece	Total	42.5	24.3

1244 The result of the expression:

1245 `ds_1 := round(unemployment, 0) or round(dataset=unemployment, decimals=0)`

1246 will be:

ds_1					
AGE	TIME	GEO	SEX	YOUTH_UNEMPLOYMENT	UNEMPLOYMENT
From 20 to 29 years	2011	Germany	Total	8	6
From 20 to 29 years	2012	Germany	Total	7	6
From 20 to 29 years	2011	Greece	Total	34	18
From 20 to 29 years	2012	Greece	Total	43	24

1247 As explained, the operator has been applied to all the MeasureComponents.

1248 Instead, the result of the expression:

1249 `ds_1 := round(unemployment#YOUTH_UNEMPLOYMENT, 0)`

1250

1251 will be:

AGE	TIME	GEO	SEX	YOUTH_UNEMPLOYMENT	UNEMPLOYMENT
From 20 to 29 years	2011	Germany	Total	8	5.9
From 20 to 29 years	2012	Germany	Total	7	5.5
From 20 to 29 years	2011	Greece	Total	34	17.7
From 20 to 29 years	2012	Greece	Total	43	24.3

1252

1253 **abs**

1254 **Component version**

1255 Syntax

```
1256     Component<Numeric> c :=  
1257     abs  
1258     ({component=}Component<Numeric> c_1)
```

1259 Constraints

1260 None

1261 Semantics

1262 Returns the absolute value of *c_1*.

1263 Examples

1264 Let us assume A = -5:

```
1265     B := abs(A) // B = 5
```

```
1266     C := abs(B) // C = 5
```

1267 **Dataset version**

1268 Syntax

```
1269     Dataset<?+, Component<Numeric>+> ds_1 :=  
1270     abs  
1271     ({dataset=}Dataset<?+, Component<Numeric>+> ds_2)
```

1272 Constraints

1273 - *ds_1* only has Numeric MeasureComponents (static).

1274 Semantics

1275 It takes in input a Dataset *ds_1* and returns another Dataset *ds_2* with the same
1276 IdentifierComponents, AttributeComponents (according to attribute propagation rules) and
1277 the same MeasureComponents. For each row in *ds_1* there is a row in *ds_2* where the
1278 MeasureComponents contain the absolute values of the corresponding ones in *ds_1*.

1279 Parameters

1280 *ds_1* - the input Dataset

1281 Returns

1282 *ds_2* - the output Dataset

1283 Example

Dataset A			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	0.484183
FR	Females	2011	-0.515817
FR	Total	2011	-1.000000

1284 The following expression

1285 DatasetB := abs (DatasetA)

1286 applied on DatasetA, will return the result:

Dataset B			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	0.484183
FR	Females	2011	0.515817
FR	Total	2011	1.000000

1287

1288 **trunc**

1289 **Component version**

1290 Syntax

1291 Component<Numeric> c :=
1292 **trunc**
1293 ({**component=**}Component<Numeric> c_1, {**decimals=**}Constant<Integer>
1294 decimals)

1295 Constraints

1296 None

1297 Semantics

1298 Discards the decimal digits of *c_1*, beyond the number of digits specified by decimals.

1299 Examples

1300 Let us assume P = 3.14159

1301 A := trunc (P, 2) // A = 3.14

1302 B := trunc (P, 4) // B = 3.1415

1303 Dataset version

1304 Syntax

```
1305 Dataset<?+, MeasureComponent<Numeric>+> ds_2 :=  
1306 trunc  
1307 ({dataset=}Dataset<?+, MeasureComponent<Numeric>+> ds_1,  
1308 {decimals=}Constant<Integer> decimals)
```

1309 Constraints

- 1310 - *ds_1* only has Numeric MeasureComponents (static).
- 1311 - *decimals* >0 (dynamic).

1312 Semantics

1313 It takes in input a Dataset *ds_1* and returns another Dataset *ds_2* with the same
1314 IdentifierComponents , AttributeComponents (according to attribute propagation rules) and
1315 the same MeasureComponents. For each row in *ds_1* there is a row in *ds_2* where the
1316 MeasureComponents are obtained by discarding the decimal digits after the *decimals* position.

1317 Parameters

1318 *ds_1* - the input Dataset

1319 Returns

1320 *ds_2* - the output Dataset

1321 Example

DatasetA			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	0.484183
FR	Females	2011	0.515817
FR	Total	2011	1.000000

1322 The result of the expression:

```
1323 ds_1 := trunc(DatasetA, 2)
```

1324 will be:

COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	0.48
FR	Females	2011	0.51
FR	Total	2011	1.00

1325 **exp**

1326 **Component version**

1327 Syntax

```
1328     Component<Numeric> c :=  
1329     exp  
1330     ({component=}Component<Numeric> c_1)
```

1331 Constraints

1332 None

1333 Semantics

1334 Returns e (Napier's – or Euler's – constant) raised to c_1 .

1335 Examples

1336 If B = 5:

```
1337     A := exp(B) // A = 148.413
```

1338 If B = -1:

```
1339     A := exp(B) // A = 0.368
```

1340 **Dataset version**

1341 Syntax

```
1342     Dataset<?+, Component<Numeric>+> ds_2 :=  
1343     exp  
1344     ({dataset=}Dataset<?+, Component<Numeric>+> ds_1)
```

1345 Constraints

1346 - ds_1 only has Numeric MeasureComponents (static).

1347 Semantics

1348 It takes in input a Dataset ds_1 and returns another Dataset ds_2 with the same
1349 IdentifierComponents, AttributeComponents (according to attribute propagation rules) and
1350 the same MeasureComponents. For each row in ds_1 there is a row in ds_2 where the
1351 MeasureComponents are obtained by elevating e (Nepero's number) to the value in the
1352 original MeasureComponent.

1353 Parameters

1354 ds_1 – the input Dataset

1355 Returns

1356 ds_2 – the output Dataset

1357 Example

Dataset A			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	5
FR	Females	2011	8
FR	Total	2011	2

1358 The following expression

1359 DatasetB := exp(DatasetA)

1360 applied on DatasetA, will return the result:

Dataset B			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	148.41
FR	Females	2011	2980.95
FR	Total	2011	7.389

1361

1362 **ln**

1363 **Component version**

1364 Syntax

1365 Component<Numeric> c :=
1366 **ln**
1367 ({**component=**}Component<Numeric> c_1)

1368 Constraints

1369 $c_1 > 0$ (dynamic).

1370 Semantics

1371 Returns the natural logarithm (base e) of c_1 .

1372 Examples

1373 If B = 1:

1374 A := ln(B) // A = 0

1375 If B = 148:

1376 A := ln(B) // A = 4.997

1377 **Dataset version**

1378 Syntax

1379 Dataset<?+, MeasureComponent<Numeric>+> *ds_2* :=
1380 **ln**(**{dataset=}**Dataset<?+, MeasureComponent<Numeric>+> *ds_1*)

1381 Constraints

1382 - *ds_1* only has Numeric MeasureComponents (static).

1383 Semantics

1384 It takes in input a Dataset *ds_1* and returns another Dataset *ds_2* with the same
1385 IdentifierComponents, AttributeComponents (according to attribute propagation rules) and
1386 the same MeasureComponents. For each row in *ds_1* there is a row in *ds_2* where the
1387 MeasureComponents are obtained by calculating the natural logarithm (in base *e*, Nepero's
1388 number) of the value in the original MeasureComponent.

1389 Parameters

1390 *ds_1* – the input Dataset

1391 Returns

1392 *ds_2* – the output Dataset

1393 Example

Dataset A			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	148.41
FR	Females	2011	2980.95
FR	Total	2011	7.389

1394

1395 The following expression

1396 DatasetB := ln(DatasetA)

1397 applied to DatasetA, will return the result:

Dataset B			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	5
FR	Females	2011	8
FR	Total	2011	2

1398 **log**

1399 **Component version**

1400 Syntax

```
1401     Component<Numeric> c :=  
1402     log  
1403     ({component=}Component<Numeric> c_1, {base=}Constant<Integer> base)
```

1404 Constraints

1405 $c_1 > 0$ (dynamic).

1406 $base > 0$ (static).

1407 Semantics

1408 Returns the base b logarithm of c_1 .

1409 Examples

1410 If $B = 1024$:

```
1411     A := log(B, 2) // A = 10
```

```
1412     A := log(B, 10) // A = 3.01
```

1413 **Dataset version**

1414 Syntax

```
1415     Dataset<?+, MeasureComponent<Numeric>+> ds_2 :=  
1416     log ({dataset=}Dataset<?+, MeasureComponent<Numeric>+> ds_1,  
1417     {base=}Constant<Numeric> base)
```

1418 Constraints

1419 - ds_1 only has Numeric MeasureComponents (static).

1420 - $base > 0$ (static)

1421 - ds_1 Measure values > 0 (dynamic)

1422 Semantics

1423 It takes in input a Dataset ds_1 and a Numeric Constant and returns another Dataset ds_2 with
1424 the same IdentifierComponents, AttributeComponents (according to attribute propagation
1425 rules) and the same MeasureComponents. For each row in ds_1 there is a row in ds_2 where
1426 the MeasureComponents are obtained by calculating the logarithm in base $base$ of the value in
1427 the original MeasureComponent.

1428 Parameters

1429 ds_1 – the input Dataset

1430 $base$ – the logarithm base

1431 Returns

1432 *ds_2* – the output Dataset

1433 Example

Dataset A			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	1024
FR	Females	2011	64
FR	Total	2011	32

1434

1435 The following expression

1436 `DatasetB := log(2, DatasetA)`

1437 applied to DatasetA, will return the result:

1438

Dataset B			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	10
FR	Females	2011	6
FR	Total	2011	5

1439 **power**

1440 **Component version**

1441 Syntax

1442 `Component<Numeric> c :=`

1443 **power**

1444 `({component=}Component<Numeric> base, {exponent=}Component<Integer>`
1445 `exponent)`

1446 Constraints

1447 None

1448 Semantics

1449 Returns base raised to the exponent power.

1450 Example

1451 If A = 2, B = 5:

1452 `C := power(B, A) // C = 25`

1453 **Dataset version**

1454 Syntax

1455 `ds_2 :=`
1456 `power({dataset=}Dataset<?+, MeasureComponent<Numeric>+> ds_1,`
1457 `{exponent=}Constant<Numeric> exponent)`

1458 Constraints

1459 - *ds_1* only has Numeric MeasureComponents (static).

1460 Semantics

1461 It takes in input a Dataset *ds_1* and a Numeric Constant and returns another Dataset *ds_2* with
1462 the same IdentifierComponents, AttributeComponents (according to attribute propagation
1463 rules) and the same MeasureComponents. For each row in *ds_1* there is a row in *ds_2* where
1464 the MeasureComponents are obtained by elevating the original MeasureComponent to the
1465 exponent-th power.

1466 Parameters

1467 *ds_1* – the input Dataset

1468 *exponent* – the power exponent

1469 Returns

1470 *ds_2* – the output Dataset

1471 Example

Dataset A			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	3
FR	Females	2011	4
FR	Total	2011	5

1472

1473 The following expression

1474 `DatasetB := power(DatasetA, 2)`

1475 applied to DatasetA, will return the result:

1476

Dataset B			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	9
FR	Females	2011	16
FR	Total	2011	25

1477 **nroot**

1478 **Component version**

1479 Syntax

1480 Component<Numeric> c :=

1481 **nroot**

1482 ({**component=**}Component<Numeric> c_1, {**index=**}Component<Integer>

1483 index)

1484 Constraints

1485 $c_1 \geq 0$ when index even (dynamic).

1486 Semantics

1487 Returns the n-th root of c_1

1488 Example

1489 If A = 2, B = 25:

1490 C := nroot(B, A) // C = 5

1491 **Dataset version**

1492 Syntax

1493 Dataset<?+, MeasureComponent<Numeric>+> ds_2 :=

1494 **nroot**

1495 ({**dataset=**}Dataset<?+, MeasureComponent<Numeric>+> ds_1,

1496 {**index=**}Constant<Numeric> n)

1497 Constraints

1498 - ds_1 only has Numeric MeasureComponents (static).

1499 - ds_1 Measure value is positive when n is even (dynamic).

1500 Semantics

1501 It takes in input a Dataset ds_1 and a Numeric Constant and returns another Dataset ds_2 with

1502 the same IdentifierComponents, AttributeComponents (according to attribute propagation

1503 rules) and the same MeasureComponents. For each row in ds_1 there is a row in ds_2 where

1504 the MeasureComponents are obtained by calculating the n-th root of the original
1505 MeasureComponent.

1506 Parameters

1507 *ds_1* – the input Dataset

1508 *n* – the index of the root

1509 Returns

1510 *ds_2* – the output Dataset

1511

1512 Example

Dataset A			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	8
FR	Females	2011	27
FR	Total	2011	64

1513

1514 The following expression

1515 `DatasetB := nroot(DatasetA, 3)`

1516

1517 applied to DatasetA, will return the result:

1518

Dataset B			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	2
FR	Females	2011	3
FR	Total	2011	4

1519

1520 **mod**

1521 **Component version**

1522 Syntax

1523 `Component<Integer> c :=`

1524 **mod**
1525 ({**component=**}Component<Integer> c_1, {**modulo=**}Component<Integer>
1526 c_2)

1527 Constraints

1528 None

1529 Semantics

1530 Returns the remainder of the division of *c_1* by *c_2*.

1531 Example

1532 If A = 5, B = 2:

1533 C := mod(A, B) // C = 1

1534 **Dataset version**

1535 Syntax

1536 Dataset<?+, MeasureComponent<Integer>+> *ds_2* :=
1537 **mod**
1538 ({**dataset=**}Dataset<?+, MeasureComponent<Integer>+> *ds_1*,
1539 {**modulo=**}Constant<Integer> *d*)

1540 Constraints

- 1541 - *ds_1* only has Numeric MeasureComponents (static).
- 1542 - *d* > 0 (static).

1543 Semantics

1544 It takes in input a Dataset *ds_1* and a Numeric Constant *d* and returns another Dataset *ds_2*
1545 with the same IdentifierComponents, AttributeComponents (according to attribute
1546 propagation rules) and the same MeasureComponents. For each row in *ds_1* there is a row in
1547 *ds_2* where the MeasureComponents are obtained by calculating the remainder of the division
1548 of the original MeasureComponent by *d*.

1549 Parameters

1550 *ds_1* – the input Dataset

1551 *d* – the divisor

1552 Returns

1553 *ds_2* – the output Dataset

1554

1555

1556

1557 Example

1558

Dataset A			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	7
FR	Females	2011	10
FR	Total	2011	12

1559 The following expression

1560 `DatasetB := mod(DatasetA, 3)`

1561 applied on DatasetA, will return the result:

Dataset B			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	1
FR	Females	2011	1
FR	Total	2011	0

1562

1563 Boolean operators

1564 VTL provides the three basic Boolean operators that allow calculations on Boolean
1565 Components.

1566 and

1567 Component version

1568 Syntax

```
1569     Component<Boolean> c :=  
1570     Component<Boolean> c_1  
1571     and  
1572     Component<Boolean> c_2
```

1573 Constraints

1574 None

1575 Semantics

1576 Returns the logical and between *c_1* and *c_2*.

1577 Dataset version

1578 Syntax

```
1579     Dataset<?+, MeasureComponent<Boolean>+> ds_3 :=  
1580     [Dataset<?+, MeasureComponent<Boolean>+> | Constant<Boolean>] ds_1  
1581     and  
1582     [Dataset<?+, MeasureComponent<Boolean>+> | Constant<Boolean>] ds_2
```

1583 Constraints

- 1584 - at least one between *ds_1* and *ds_2* must be a Dataset (static).
- 1585 - if *ds_1* (*ds_2*) is a Dataset, it must have only Boolean MeasureComponents (static).
- 1586 - If both *ds_1* and *ds_2* are Datasets, they must have the same IdentifierComponents in
1587 name and type or, at least, the IdentifierComponents of *ds_1* must be a subset of the
1588 ones of *ds_2* (or viceversa) (static).
- 1589 - If both *ds_1* and *ds_2* are Datasets, they must have the same MeasureComponents
1590 (static).
- 1591 - In case *ds_1* (*ds_2*) is a Constant<Boolean>, *ds_2* (*ds_1*) must have only one
1592 MeasureComponent<Boolean> (static).

1593 Semantics

1594 The operator performs a logical conjunction on the operand Datasets or constants.

1595 It takes in input two Datasets *ds_1* and *ds_2* with the same IdentifierComponents, or at least
1596 the identifier components of one being a subset of the IdentifierComponents of the other, and

1597 the same MeasureComponents (in name and with Boolean type). It returns a third Dataset
1598 *ds_3*, having the superset of the IdentifierComponents of *ds_1* and *ds_2*, some
1599 AttributeComponents (according to the attribute propagation rules) and, for each pair of
1600 homonym MeasureComponents of *ds_1* and *ds_2* a MeasureComponent<Boolean> with the
1601 same name.

1602 For each pair of rows in *ds_1* and *ds_2* having the same values for all the common
1603 IdentifierComponents, the MeasureComponent<Boolean> in *ds_3* will contain the result of the
1604 logical *and* of the original values.

1605 If *ds_1* (or *ds_2*) is a Constant, *ds_3* has a single MeasureComponent<Boolean>, named as the
1606 MeasureComponent<Boolean> of *ds_2* (*ds_1*), containing the logical *and* of the original value
1607 with the Constant.

1608 Parameters

1609 *ds_1* – the first operand, Dataset or Constant of the logical operator.

1610 *ds_2* – the second operand, Dataset or Constant of the logical operator.

1611 Returns

1612 *ds_3* – the Dataset resulting from the application of the logical operator.

1613

1614 Example 1

1615 In this example we examine whether there are rows for a particular sex and age group.

1616

1617

population				
SEX	AGE	GEO	TIME	VALUE
M	Y_LT15	BE	2013	970428
M	Y15-64	BE	2013	3678355
M	Y_GE65	BE	2013	838653
F	Y_LT15	BE	2013	927644
F	Y15-64	BE	2013	3625561
F	Y_GE65	BE	2013	1121001
M	Y_LT15	UK	2013	5757444
M	Y15-64	UK	2013	20748657
M	Y_GE65	UK	2013	4917238
F	Y_LT15	UK	2013	5488356
F	Y15-64	UK	2013	20915924
F	Y_GE65	UK	2013	6068452

1618

1619

1620 The expression

1621 `population#sex="M" and population#age_group="Y15-64"`

1622

1623 will yield:

1624

SEX	AGE	GEO	TIME	CONDITION ¹
M	Y_LT15	BE	2013	false
M	Y15-64	BE	2013	true
M	Y_GE65	BE	2013	false
F	Y_LT15	BE	2013	false
F	Y15-64	BE	2013	false
F	Y_GE65	BE	2013	false
M	Y_LT15	UK	2013	false
M	Y15-64	UK	2013	true
M	Y_GE65	UK	2013	false
F	Y_LT15	UK	2013	false
F	Y15-64	UK	2013	false
F	Y_GE65	UK	2013	false

1625

1626 Each of the expressions `population#sex="M"` and `population#age_group="Y15-`
1627 `64"` specifies implicitly a Boolean measure that shows whether the condition stated in
1628 the expression holds. The and operator creates the conjunction of these two measures.

1629 **or**

1630 **Component version**

1631 Syntax

1632 `Component<Boolean> c :=`

1633 `Component<Boolean> c_1`

1634 **or**

1635 `Component<Boolean> c_2`

1636 Constraints

1637 None

¹ This column is named `CONDITION`, since it is essentially a result of a Boolean operation on two comparison operations.

1638 Semantics

1639 Returns the logical or between *c_1* and *c_2*.

1640 **Dataset version**

1641 Syntax

```
1642 Dataset<?+, MeasureComponent<Boolean>+> ds_3 :=  
1643 [Dataset<?+, MeasureComponent<Boolean>+> | Constant<Boolean>] ds_1  
1644 or  
1645 [Dataset<?+, MeasureComponent<Boolean>+> | Constant<Boolean>] ds_2
```

1646 Constraints

- 1647 - at least one between *ds_1* and *ds_2* must be a Dataset (static).
- 1648 - if *ds_1* (*ds_2*) is a Dataset, it must have only Boolean MeasureComponents (static).
- 1649 - If both *ds_1* and *ds_2* are Datasets, they must have the same IdentifierComponents in
1650 name and type or, at least, the IdentifierComponents of *ds_1* must be a subset of the
1651 ones of *ds_2* (or vice-versa) (static).
- 1652 - If both *ds_1* and *ds_2* are Datasets, they must have the same MeasureComponents
1653 (static).
- 1654 - In case *ds_1* (*ds_2*) is a Constant<Boolean>, *ds_2* (*ds_1*) must have only one
1655 MeasureComponent<Boolean> (static).

1656 Semantics

1657 The operator performs a logical disjunction on the operand Datasets or constants.

1658 It takes in input two Datasets *ds_1* and *ds_2* with the same IdentifierComponents, or at least
1659 the identifier components of one being a subset of the IdentifierComponents of the other, and
1660 the same MeasureComponents (in name and with Boolean type). It returns a third Dataset
1661 *ds_3*, having the superset of the IdentifierComponents of *ds_1* and *ds_2*, some
1662 AttributeComponents (according to the attribute propagation rules) and, for each pair of
1663 homonym MeasureComponents of *ds_1* and *ds_2* a MeasureComponent<Boolean> with the
1664 same name.

1665 For each pair of rows in *ds_1* and *ds_2* having the same values for all the common
1666 IdentifierComponents, the MeasureComponent<Boolean> in *ds_3* will contain the result of the
1667 logical *or* of the original values.

1668 If *ds_1* (or *ds_2*) is a Constant, *ds_3* has a single MeasureComponent<Boolean>, named as the
1669 MeasureComponent<Boolean> of *ds_2* (*ds_1*), containing the logical *or* of the original value
1670 with the Constant.

1671 Parameters

1672 *ds_1* – the first operand, Dataset or Constant of the logical operator.

1673 *ds_2* – the second operand, Dataset or Constant of the logical operator.

1674 Returns

1675 *ds_3* – the Dataset resulting from the application of the logical operator.

1676 Example

1677 In this example we examine whether there are rows for a particular sex or age group.

1678 Consider the following Dataset of population.

1679

1680

population				
SEX	AGE	GEO	TIME	VALUE
M	Y_LT15	BE	2013	970428
M	Y15-64	BE	2013	3678355
M	Y_GE65	BE	2013	838653
F	Y_LT15	BE	2013	927644
F	Y15-64	BE	2013	3625561
F	Y_GE65	BE	2013	1121001
M	Y_LT15	UK	2013	5757444
M	Y15-64	UK	2013	20748657
M	Y_GE65	UK	2013	4917238
F	Y_LT15	UK	2013	5488356
F	Y15-64	UK	2013	20915924
F	Y_GE65	UK	2013	6068452

1681

1682 The expression

1683 `population#sex="M" OR population#age_group="Y15-64"`

1684 will yield:

1685

SEX	AGE	GEO	TIME	VALUE
M	Y_LT15	BE	2013	true
M	Y15-64	BE	2013	true
M	Y_GE65	BE	2013	true
F	Y_LT15	BE	2013	false
F	Y15-64	BE	2013	true
F	Y_GE65	BE	2013	false
M	Y_LT15	UK	2013	true
M	Y15-64	UK	2013	true
M	Y_GE65	UK	2013	true
F	Y_LT15	UK	2013	false
F	Y15-64	UK	2013	true
F	Y_GE65	UK	2013	false

1686

1687 XOR

1688 Component version

1689 Syntax

```
1690     Component<Boolean> c :=  
1691     Component<Boolean> c_1  
1692     xor  
1693     Component<Boolean> c_2
```

1694 Constraints

1695 None

1696 Semantics

1697 Returns the logical xor between *c_1* and *c_2*.

1698 Dataset version

1699 Syntax

```
1700     Dataset<?+, MeasureComponent<Boolean>+> ds_3 :=  
1701     [Dataset<?+, MeasureComponent<Boolean>+> | Constant<Boolean>] ds_1  
1702     xor  
1703     [Dataset<?+, MeasureComponent<Boolean>+> | Constant<Boolean>] ds_2
```

1704 Constraints

- 1705 - at least one between *ds_1* and *ds_2* must be a Dataset (static).
- 1706 - if *ds_1* (*ds_2*) is a Dataset, it must have only Boolean MeasureComponents (static).
- 1707 - If both *ds_1* and *ds_2* are Datasets, they must have the same IdentifierComponents in
1708 name and type or, at least, the IdentifierComponents of *ds_1* must be a subset of the
1709 ones of *ds_2* (or vice-versa) (static).
- 1710 - If both *ds_1* and *ds_2* are Datasets, they must have the same MeasureComponents
1711 (static).
- 1712 - In case *ds_1* (*ds_2*) is a Constant<Boolean>, *ds_2* (*ds_1*) must have only one
1713 MeasureComponent<Boolean> (static).

1714 Semantics

1715 The operator performs a logical exclusive disjunction on the operand Datasets or constants.

1716 It takes in input two Datasets *ds_1* and *ds_2* with the same IdentifierComponents, or at least
1717 the identifier components of one being a subset of the IdentifierComponents of the other, and
1718 the same MeasureComponents (in name and with Boolean type).

1719 It returns a third Dataset *ds_3*, having the superset of the IdentifierComponents of *ds_1* and
1720 *ds_2*, some AttributeComponents (according to the attribute propagation rules) and, for each
1721 pair of homonym MeasureComponents of *ds_1* and *ds_2* a MeasureComponent<Boolean> with
1722 the same name.

1723 For each pair of rows in *ds_1* and *ds_2* having the same values for all the common
1724 IdentifierComponents, the MeasureComponent<Boolean> in *ds_3* will contain the result of the
1725 logical *xor* of the original values.

1726 If *ds_1* (or *ds_2*) is a Constant, *ds_3* has a single MeasureComponent<Boolean>, named as the
1727 MeasureComponent<Boolean> of *ds_2* (*ds_1*), containing the logical *xor* of the original value
1728 with the Constant.

1729 Parameters

1730 *ds_1* – the first operand, Dataset or Constant of the logical operator.

1731 *ds_2* – the second operand, Dataset or Constant of the logical operator.

1732 Returns

1733 *ds_3* – the Dataset resulting from the application of the logical operator.

1734

1735 Example

1736 In this example we examine whether there are rows that correspond to only one of a
1737 particular sex and a particular age group.

1738 Consider the following Dataset of population.

1739

1740

population				
SEX	AGE	GEO	TIME	VALUE
M	Y_LT15	BE	2013	970428
M	Y15-64	BE	2013	3678355
M	Y_GE65	BE	2013	838653
F	Y_LT15	BE	2013	927644
F	Y15-64	BE	2013	3625561
F	Y_GE65	BE	2013	1121001
M	Y_LT15	UK	2013	5757444
M	Y15-64	UK	2013	20748657
M	Y_GE65	UK	2013	4917238
F	Y_LT15	UK	2013	5488356
F	Y15-64	UK	2013	20915924
F	Y_GE65	UK	2013	6068452

1741

1742 The expression

1743 `population#sex="M" xor population#age_group="Y15-64"`

1744

1745

1746

1747

1748 will yield:

1749

SEX	AGE	GEO	TIME	VALUE
M	Y_LT15	BE	2013	true
M	Y15-64	BE	2013	false
M	Y_GE65	BE	2013	true
F	Y_LT15	BE	2013	false
F	Y15-64	BE	2013	true
F	Y_GE65	BE	2013	false
M	Y_LT15	UK	2013	true
M	Y15-64	UK	2013	false
M	Y_GE65	UK	2013	true
F	Y_LT15	UK	2013	false
F	Y15-64	UK	2013	true
F	Y_GE65	UK	2013	false

1750

1751 **not**

1752 **Component version**

1753 Syntax

1754 Component<Boolean> c :=

1755 **not**

1756 Component<Boolean> c_1

1757 Constraints

1758 None

1759 Semantics

1760 Returns the logical negation of *c_1*.

1761 **Dataset version**

1762 Syntax

1763 Dataset<?+, MeasureComponent<Boolean>+> ds_2 :=

1764 **not**

1765 Dataset<?+, MeasureComponent<Boolean>+> ds_1

1766 Constraints

1767 - if *ds_1* must have only Boolean MeasureComponents (static).

1768 Semantics

1769 Takes in input a Dataset *ds_1* and returns a new Dataset *ds_2* with the same
1770 IdentifierComponents, the AttributeComponents (according to attribute propagation rules)
1771 and the same MeasureComponents. For each row in *ds_1* there is a row in *ds_2* with the
1772 logical negation (*not*) of the original Boolean value.

1773 Parameters

1774 *ds_1* – the input Dataset .

1775 Returns

1776 *ds_2* – the output Dataset.

1777

1778 Example

1779 In this example we examine whether there are rows that do not correspond to a particular
1780 sex.

1781 Consider the following Dataset of population.

1782

1783

population				
SEX	AGE	GEO	TIME	VALUE
M	Y_LT15	BE	2013	970428
M	Y15-64	BE	2013	3678355
M	Y_GE65	BE	2013	838653
F	Y_LT15	BE	2013	927644
F	Y15-64	BE	2013	3625561
F	Y_GE65	BE	2013	1121001
M	Y_LT15	UK	2013	5757444
M	Y15-64	UK	2013	20748657
M	Y_GE65	UK	2013	4917238
F	Y_LT15	UK	2013	5488356
F	Y15-64	UK	2013	20915924
F	Y_GE65	UK	2013	6068452

1784

1785 The expression

1786 `not population#sex="M"`

1787

1788

1789

1790

1791

will yield:

SEX	AGE	GEO	TIME	VALUE
M	Y_LT15	BE	2013	false
M	Y15-64	BE	2013	false
M	Y_GE65	BE	2013	false
F	Y_LT15	BE	2013	true
F	Y15-64	BE	2013	true
F	Y_GE65	BE	2013	true
M	Y_LT15	UK	2013	false
M	Y15-64	UK	2013	false
M	Y_GE65	UK	2013	false
F	Y_LT15	UK	2013	true
F	Y15-64	UK	2013	true
F	Y_GE65	UK	2013	true

1792 Relational operations

1793 union (records)

1794 Syntax

```
1795     Dataset<?> ds_2 :=  
1796     union  
1797     (Dataset<?> ds_1 {, Dataset<?> ds_1}*)
```

1798 Constraints

1799 - all the Datasets *ds_1* must have the same IdentifierComponents and MeasureComponents, in
1800 name and type (static).

1801 Semantics

1802 Performs row-wise union. Takes in input one or more Datasets *ds_1* with the same
1803 IdentifierComponents and MeasureComponents and returns a new Dataset *ds_2* with the
1804 same IdentifierComponents and MeasureComponents and AttributeComponents (according
1805 to attribute propagation rules).

1806 The result contains all the rows from every *ds_1* without duplicates, which is their union.
1807 Therefore, in the output there cannot be two rows with the same values for all the
1808 IdentifierComponents and MeasureComponents. If only one argument is given, it is returned
1809 unchanged.

1810 It is worth underlying that, on the other hand, it is possible to have in *ds_2* two rows with the
1811 same values for the IdentifierComponents. However, if this happens, this will result in a
1812 runtime error if *ds_2* is actually assigned in an expression as Datasets with duplicates are only
1813 admitted in will not be allowed to be stored into the right-hand side.

1814 Parameters

1815 *ds_1* - all the input Datasets to consider in the union.

1816 Returns

1817 *ds_2* - the output Dataset

1818

1819 Example 1

1820 In this example we perform a union of two Datasets, with the same IdentifierComponents and
1821 MeasureComponents.

1822

1823 Both Datasets contain information about the total population of a set of countries.

1824

total_population1

TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	5
2012	Greece	Total	Total	2
2012	France	Total	Total	3
2012	Malta	Total	Total	7
2012	Finland	Total	Total	9
2012	Switzerland	Total	Total	12

1825

Total_population2				
TIME	GEO	AGE	SEX	POPULATION
2012	Netherlands	Total	Total	23
2012	Greece	Total	Total	2
2012	Spain	Total	Total	5
2012	Iceland	Total	Total	1

1826

1827

The result of the expression

1828

`ds_r := union(total_population1, total_population2)`

1829

will be:

ds_r				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	5
2012	Greece	Total	Total	2
2012	France	Total	Total	3
2012	Malta	Total	Total	7
2012	Finland	Total	Total	9
2012	Switzerland	Total	Total	12
2012	Netherlands	Total	Total	23
2012	Spain	Total	Total	5
2012	Iceland	Total	Total	1

1830

1831 Example 2

1832 In this example we perform a union of two Datasets that contain population data for a group
 1833 of countries for two different reference years.

1834

total_population1				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	1
2012	Greece	Total	Total	2
2012	France	Total	Total	3
2012	Malta	Total	Total	4
2012	Finland	Total	Total	5
2012	Switzerland	Total	Total	6

1835

total_population2				
TIME	GEO	AGE	SEX	POPULATION
2011	Belgium	Total	Total	10
2011	Greece	Total	Total	20
2011	France	Total	Total	30
2011	Malta	Total	Total	40
2011	Finland	Total	Total	50
2011	Switzerland	Total	Total	60

1836

The result of the expression

1837

`ds_r := union(total_population1, total_population2)`

1838

will be:

ds_r				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	1
2012	Greece	Total	Total	2
2012	France	Total	Total	3
2012	Malta	Total	Total	4
2012	Finland	Total	Total	5
2012	Switzerland	Total	Total	6
2011	Belgium	Total	Total	10
2011	Greece	Total	Total	20
2011	France	Total	Total	30
2011	Malta	Total	Total	40

2011	Finland	Total	Total	50
2011	Switzerland	Total	Total	60

1839

1840 intersect

1841 Syntax

```
1842 Dataset<?> ds_2 :=
1843 intersect
1844 (Dataset<?> ds_1 {, Dataset<?> ds_1}*)
```

1845 Constraints

1846 - all the Datasets *ds_1* must have the same IdentifierComponents and MeasureComponents, in
1847 name and type (static).

1848 Semantics

1849 Performs row-wise intersection. Takes in input one or more Datasets *ds_1* with the same
1850 IdentifierComponents and MeasureComponents and returns a new Dataset *ds_2* with the
1851 same IdentifierComponents and MeasureComponents and AttributeComponents (according
1852 to attribute propagation rules).

1853 The result only contains the rows that are present in every *ds_1*, which is their intersection. If
1854 only one argument is given, it is returned unchanged. For the comparison, both the
1855 IdentifierComponents and the MeasureComponents values are considered.

1856 Parameters

1857 *ds_1* - all the input Datasets to consider in the intersection.

1858 Returns

1859 *ds_2* - the output Dataset

1860

1861 Example

1862 In this example, we perform an intersection of the Datasets *total_population1*,
1863 *total_population2*.

1864 The result will contain information of the total population of the set of countries that are
1865 common in the initial datasets.

1866

total_population1				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	1
2012	Greece	Total	Total	2

2012	France	Total	Total	3
2012	Malta	Total	Total	4
2012	Finland	Total	Total	5
2012	Switzerland	Total	Total	6

1867

total_population2				
TIME	GEO	AGE	SEX	POPULATION
2011	Belgium	Total	Total	10
2012	Greece	Total	Total	2
2011	France	Total	Total	30
2011	Malta	Total	Total	40
2011	Finland	Total	Total	50
2011	Switzerland	Total	Total	60

1868

The expression

1869

```
d_r := intersect(total_population1, total_population2)
```

1870

will yield the result:

d_r				
TIME	GEO	AGE	SEX	POPULATION
2012	Greece	Total	Total	2

1871

1872 **symdiff**

1873 Syntax

1874

```
Dataset<?> ds_3 :=
```

1875

```
symdiff
```

1876

```
(({dataset1=}Dataset<?> ds_1, {dataset2=}Dataset<?> ds_2)
```

1877 Constraints

1878

- *ds_1* and *ds_2* must have the same IdentifierComponents and MeasureComponents, in name

1879

and type (static).

1880 Semantics

1881

Performs row-wise symmetric difference. It takes in input two Datasets *ds_1* and *ds_2* with

1882

the same IdentifierComponents and MeasureComponents and returns a new Dataset *ds_3*

1883

with the same IdentifierComponents and MeasureComponents and AttributeComponents

1884

(according to attribute propagation rules). The result contains only the rows that are present

1885 either *ds_1* or in *ds_2* but not in both, which is their symmetric difference. For the comparison,
1886 both the IdentifierComponents and the MeasureComponents values are considered.

1887 Parameters

1888 *ds_1* – the first input Dataset to consider in the symmetric difference.

1889 *ds_2* – the second input Dataset to consider in the symmetric difference.

1890 Returns

1891 *ds_3* – the output Dataset

1892 Examples

1893 In this example we perform a symmetric difference of the Datasets *total_population1*,
1894 *total_population2*. The result will contain information of the total population of the countries
1895 that are not common in the initial datasets.

1896

total_population1				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	1
2012	Greece	Total	Total	2
2012	France	Total	Total	3
2012	Malta	Total	Total	4
2012	Finland	Total	Total	5
2012	Switzerland	Total	Total	6

1897

total_population2				
TIME	GEO	AGE	SEX	POPULATION
2011	Belgium	Total	Total	1
2012	Greece	Total	Total	2
2012	France	Total	Total	3
2012	Malta	Total	Total	4
2012	Finland	Total	Total	5
2012	Switzerland	Total	Total	6

1898

1899 The expression:

1900 `d_r := symdiff(total_population1, total_population2)`

1901

1902 will yield the result:

d_r				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	1
2011	Belgium	Total	Total	1

1903

1904 **setdiff**

1905 Syntax

```
1906 Dataset<?> ds_3 :=  
1907 Dataset<?> ds_1  
1908 setdiff  
1909 Dataset<?> ds_2
```

1910 Constraints

1911 - *ds_1* and *ds_2* must have the same IdentifierComponents and MeasureComponents, in name
1912 and type (static).

1913 Semantics

1914 Performs row-wise difference. It takes as input two Datasets *ds_1* and *ds_2* with the same
1915 IdentifierComponents and MeasureComponents and returns a new Dataset *ds_3* with the
1916 same IdentifierComponents and MeasureComponents and AttributeComponents (according
1917 to attribute propagation rules).

1918 The result only contains the rows that are present in *ds_1* but not in *ds_2*, which is their
1919 difference. For the comparison, both the IdentifierComponents and the MeasureComponents
1920 values are considered.

1921 Parameters

1922 *ds_1* - the first input Dataset to consider in the difference.

1923 *ds_2* - the second input Dataset to consider in the difference.

1924 Returns

1925 *ds_3* - the output Dataset

1926

1927 Example 1

1928 In this example we perform a difference between two Datasets that contain population data
1929 for a group of countries for two different reference years.

1930

1931

total_population1				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	10
2012	Greece	Total	Total	20
2012	France	Total	Total	30
2012	Malta	Total	Total	40
2012	Finland	Total	Total	50
2012	Switzerland	Total	Total	60

1932

total_population2				
TIME	GEO	AGE	SEX	POPULATION
2011	Belgium	Total	Total	10
2012	Greece	Total	Total	20
2012	France	Total	Total	30
2012	Malta	Total	Total	40
2012	Finland	Total	Total	50
2012	Switzerland	Total	Total	60

1933

The statement

1934

`d_r := total_population1 setdiff total_population2`

1935

will yield the result:

d_r				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	10

1936

1937 Example 2

Dataset A			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	7
FR	Females	2011	10
FR	Total	2011	12

1938

Dataset B			
COUNTRY	SEX	YEAR	VALUE
FR	Males	2011	7
FR	Females	2011	10

1939 The following expression
 1940 `DatasetC := DatasetA setdiff DatasetB`
 1941 applied on DatasetA and DatasetB will return the result:

Dataset C			
COUNTRY	SEX	YEAR	VALUE
FR	Total	2011	12

1942

1943 merge

1944 Syntax

```

1945 Dataset<?> ds_3 :=
1946 merge
1947 (Dataset<?> ds_1 {as} Constant<String> alias_1,
1948 Dataset<?> ds_2 {as} Constant<String> alias_2
1949 {, Dataset<?> ds_2 {as} Constant<String> alias_2}*)
1950 on (Component<Boolean> cond)
1951 return (
1952 {Component<?> comp {as} Constant<String> alias_3}
1953 {, Component<?> comp {as} Constant<String>}* alias_3))

```

1954 Constraints

- 1955 - *cond* is a Component expression evaluating to a Boolean. Literals (name of
- 1956 Components) must be qualified by prefixing them with the alias name (*alias_1* or
- 1957 *alias_2*) (static).
- 1958 - *aliases* must be alphanumeric Strings corresponding to valid names for Components
- 1959 (static).
- 1960 - *Comp* is a Component expression where only the names of Components from *ds_1* or
- 1961 *ds_2* are allowed. They must be prefixed with the alias name (*alias_1* or *alias_2*) as it
- 1962 guarantees no ambiguities (static).
- 1963 - *alias_1* and {*alias_2*}+ must be different (static).
- 1964 - all the {*alias_3*}+ must be different as there cannot be duplicated Component names in
- 1965 the output (static).

1966 Semantics

1967 Takes in input two or more Datasets *ds_1*, *ds_2* and returns a third Dataset *ds_3*, which is
1968 obtained as follows:

- 1969 - *ds_1* and *ds_2* are given an alias *ds_1* and *ds_2*, respectively
- 1970 - the Cartesian product of all the rows from *ds_1* and *ds_2* is temporarily calculated
- 1971 - from the Cartesian products, the rows for which the Boolean Component expression
1972 evaluates to *true* are kept, while the others are filtered out. The Component<Boolean>
1973 is a qualified expression involving *ds_1* and *ds_2* components (e.g. *alias_1#k1* =
1974 *alias_2#k2* and *alias_1#k3* > *alias_1#k4*)
- 1975 - from the kept rows in the Cartesian products, the output rows are produced by
1976 keeping the only Components specified in *comp* Component expressions. For each of
1977 them, the role (i.e. if they are IdentifierComponent or MeasureComponent) is directly
1978 derived from the role they have in *ds_1* or *ds_2* and an *alias_3*, which is a new name,
1979 must be assigned to each of them.
- 1980 - AttributeComponents are added according to the attribute propagation rules.

1981 Parameters

1982 *ds_1* – the first Dataset to merge

1983 *ds_2* – all the following Dataset to merge

1984 *alias_1* – an alias for *ds_1* to be used in *cond* and *comp*

1985 *alias_2* – an alias for all the *ds_2* to be used in *cond* and *comp*

1986 *cond* – a qualified Component expression, specifying the merge conditions

1987 *comp* – a qualified Component expression (only literal Component names are allowed)
1988 specifying the desired Components in the output

1989 *alias_3* – an alias for a desired Component in the output

1990 Returns

1991 *ds_3* – the returned Dataset

1992 Example 1

1993 In this example we report the multi-measure Dataset produced by the following expression.

1994 `ds_r := merge(ds_1 "A", ds_2 "B", on(A#country =`
1995 `B#country), return(A#country as "Country" ,A#value as`
1996 `"Value",B#population as "Population"))`

1997

DS_1		
COUNTRY	CITY	VALUE
Belgium	Brussels	34
Greece	Athens	0
France	Paris	21

1998

DS_2		
COUNTRY	CITY	POPULATION
Belgium	Brussels	45
Greece	Athens	46
France	Paris	47

1999

DS_R		
COUNTRY	VALUE	POPULATION
Belgium	34	45
Greece	0	46
France	21	47

2000

2001 Example 2

2002 In this example we report the mono-measure Dataset produced by the following expression.

2003

2004 `ds_r := merge(ds_1 "A", ds_2 "B", on(A#country = B#country and A#city`
 2005 `= B#city), return(A#country as "Country", B#city as "City", A#value`
 2006 `as "Value"))`

2007

DS_R		
COUNTRY	CITY	VALUE
Belgium	Brussels	34
Greece	Athens	0
France	Paris	21

2008

2010 **min/max**2011 Syntax

```

2012 Dataset<?> ds_2 :=
2013 [min|max]
2014 ({dataset=}Dataset<?> ds_1)

```

2015 Constraints

2016 - ds_1 must have only one MeasureComponent (static).

2017 Semantics

2018 These operators find the minimum or maximum row in a Dataset according to the minimum
 2019 or maximum value of the MeasureComponent (based on numerical order for numeric values,
 2020 or lexicographical order, for String values). In case of ambiguity, the one to be returned is up
 2021 to the implementation. They take in input a Dataset *ds_1* with a single ComponentMeasure
 2022 and return a new Dataset *ds_2* with the same IdentifierComponent, MeasureComponents and
 2023 AttributeComponents (according to attribute propagation rules).

2024

2025 NULL values are not considered and rows with NULLS are never returned.

2026

2027 Example

2028 In this example we find the country with the largest (smallest) population.

2029

2030 **DatasetA**

GEO	TIME	VALUE
Sweden	2012	9.500
Denmark	2012	5.590
Finland	2012	5.400
Norway	2012	4.960
Iceland	2012	0.328
Greenland	2012	0.057
Faroe Islands	2012	0.049

2031

2032 The expression:

```
2033 max (DatasetA)
```

2034 returns:

GEO	TIME	VALUE
Sweden	2012	9.500

2035 min (DatasetA) returns:

GEO	TIME	VALUE
Faroe Islands	2012	0.049

2036 hierarchy

2037 VTL foresees the existence of set relations among Code Items in Code List.

2038 Many Enumerated Value Domains have an intrinsic Boolean algebraic structure, in the sense
2039 that a Boolean algebra can be defined on the respective Code Items.

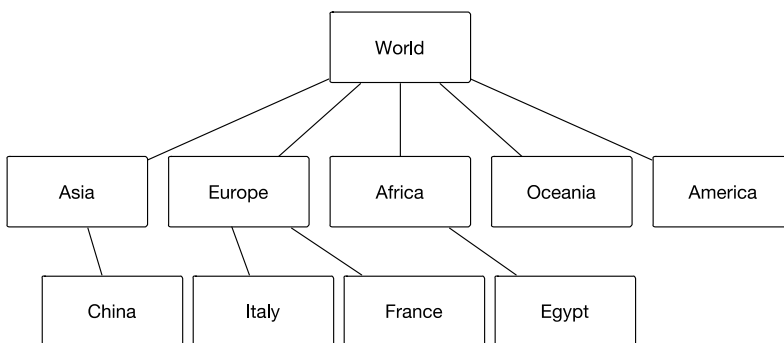
2040 In general, a Boolean algebra is an algebraic structure (elements and operators having some
2041 properties) that summarizes the properties of set operators (*union, intersection, complement*)
2042 and logical operators (or, and, not).

2043 Elements of a Boolean Value Domain can be combined with the elementary set operators²; e.g.
2044 the item C is the union of the items D and E; the item K is the complement of S with respect to
2045 J (so the elements in J that are not in S), the item A is the intersection of B and C and so on.

2046 Only considering the set union, there are two possible organizations for hierarchical Code
2047 Lists: classifications and free hierarchies.

2048 In **classifications**, every element is uniquely classified by a single partition, so that the overall
2049 structure is a tree and, consequently, every Code Item can be given a specific level. An
2050 example is the usual geographical classification of the world into continents, each partitioned
2051 into nations.

2052



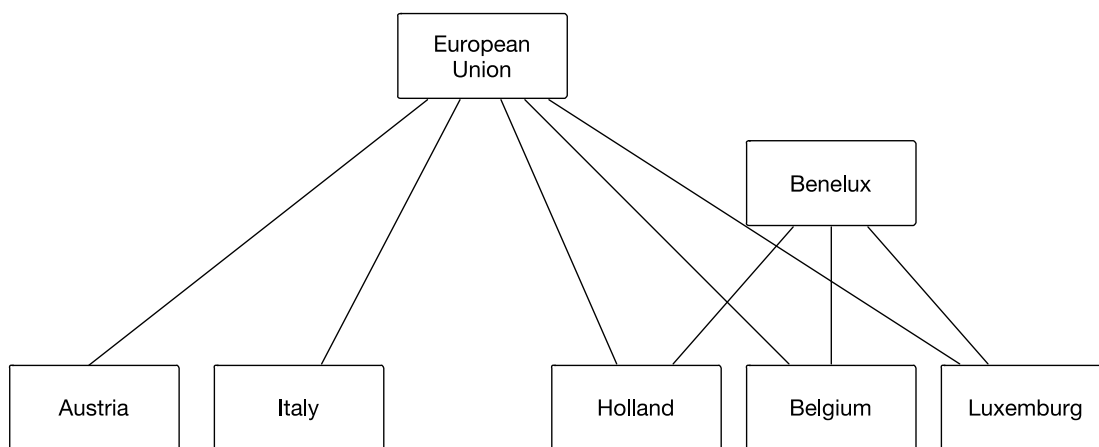
2053

2054 **Figure 1 - example of hierarchical Code List**

2055

² Here we refer to elementary set operations and not to any operator of the language.

2056 In **free hierarchies** each item can be partitioned according to multiple criteria; in turn, every
 2057 element can be used to compose multiple other elements. The overall structure is not a tree
 2058 and isolated Items can be present. An example is the hierarchy of European countries, where
 2059 Belgium, Holland and Luxembourg contribute to the “European Countries” Code Item and to
 2060 the “Benelux” Code Item.



2061
 2062 **Figure 2 - Example of free hierarchy**

2063 More sophisticated hierarchical organizations can indeed exist if intersection and
 2064 complement set operators are also considered.

2065 For example the element “Benelux” could also be defined as the complement of the European
 2066 Union with respect to all the countries except Holland, Belgium and Luxembourg.

2067 Therefore we would have:

2068
$$\text{BENELUX} = \text{EU} - (\text{ITALY} \cup \text{AUSTRIA} \cup \dots)$$

2069 In order to support multiple classifications of Code Items in Code Lists and allow for the
 2070 adoption of all the set operators, we introduce two concepts *hierarchical aggregations* and
 2071 *mappings* (that will be used in the **hierarchy** operator).

2072 A *hierarchical aggregation* is a set of *mappings*, each transforming a Code Item into another
 2073 Code Item. All the *mappings* within a *hierarchical aggregation* associate Code Items of the
 2074 same Code List with Code Items of a single Code List.

2075 Hierarchical aggregations and mappings can be easily expressed in tabular form and referred
 2076 to in the language (in **hierarchy** operator) by an identifier (*hierarchyName*). However, there
 2077 is also an inline syntactical form.

2078 Suppose we want to express the hierarchical relationship $\text{BENELUX} = \text{Belgium} \cup \text{Holland} \cup$
 2079 Luxembourg . There will be a hierarchical aggregation **Benelux_aggr**, with the following
 2080 mappings:

2081

HIERARCHY NAME	MAPS FROM	MAPS TO	START DATE	END DATE	SIGN	LEVEL	OUTPUT
Benelux_aggr	Belgium	Benelux	+	1	true
Benelux_aggr	Holland	Benelux	+	1	true
Benelux_aggr	Luxembourg	Benelux	+	1	true

2082

2083 It maps each component item into the compound one.

2084 Each mapping has a *Sign*. It specifies if the contribution of the MAPS FROM Code Item in the
2085 composition is positive (UNION) or negative (COMPLEMENT). Notice that there is not a
2086 particular convention to represent intersection, as it can be obtained with an appropriate
2087 composition of UNION and COMPLEMENT.

2088 For instance, if we want to define in the element **EuropeWithoutItaly**, a possible definition
2089 could be the one complementing Italy, with respect to the entire Europe (i.e. subtracting Italy
2090 from Europe), as shown in the following table:

2091

HIERARCHY NAME	MAPS FROM	MAPS TO	START DATE	END DATE	SIGN	LEVEL	OUTPUT
Eu_no_Italy_aggr	Europe	EuropeWithoutItaly	+	1	true
Eu_no_Italy_aggr	Italy	EuropeWithoutItaly	-	1	true

2092

2093 Moreover, mappings are divided into levels, in the sense that a complete tree can be
2094 embedded into one hierarchy. The OUTPUT property, for each MAPS TO, indicates if the value
2095 must be preserved in the output or is only to be used for aggregations at a higher level.

2096 For instance, suppose we want to express the hierarchical relationship in Figure 2, there will
2097 be a hierarchy **World_aggr**, with the following correspondences:

2098

HIERARCHY NAME	MAPS FROM	MAPS TO	START DATE	END DATE	SIGN	LEVEL	OUTPUT
World_aggr	Italy	Europe	+	1	false
World_aggr	France	Europe	+	1	false
World_aggr	Luxembourg	Europe	+	1	false
World_aggr	...	Europe			+	1	false
World_aggr	Asia	World			+	2	true
World_aggr	Europe	World			+	2	true
World_aggr	America	World			+	2	true
World_aggr	Oceania	World			+	2	true
World_aggr	Africa	World			+	2	true

2099

2100 Output false for LEVEL 1, indicates that the aggregations into Europe (and into the other
2101 countries) are only functional to LEVEL 2 and the MAPS TO values with false will not be in the
2102 output.

2103 Summarizing, the set of mappings within a hierarchy has to be interpreted as follows.

2104 For every level, from the lowest to the highest, each MAPS TO Code Item is the set difference
2105 between the UNION of all the corresponding MAPS FROM Code Items with positive SIGN and
2106 the UNION of all the corresponding MAPS FROM Code Items with the negative SIGN.

2107 *Rules* inherently represent hierarchies in Code Lists, but, at the same time, only refer to Code
2108 Items. Thus, can be applied on different *Identifier Components* referring to different Code
2109 Lists.

2110 Syntax (“hierarchy” Operator)

```
2111     Dataset ds_2 :=  
2112     hierarchy  
2113     (  
2114     {dataset=}Dataset<?,MeasureComponent<Numeric>+> ds_1,  
2115     {component=}IdentifierComponent<?> comp,  
2116     [  
2117     {hierarchy_name=}Constant<String> hierarchyname  
2118     | (  
2119     {  
2120     ( ({from=}Constant<T> maps_from, {level=}Constant<Integer> level,  
2121     {sign=} [+|-])  
2122     {, ({from=}Constant<T> maps_from, {level=}Constant<Integer> level,  
2123     {sign=} [+|-])})*  
2124     to {to=}Constant<T> maps_to}+  
2125     ) as Constant<String> hierarchyname  
2126     ], {isFilter=}Constant<Boolean> isFilter  
2127     {, {aggregation=} [sum|prod] }  
2128     )
```

2131 Constraints

- 2132 - *hierarchyname* denotes a hierarchical aggregation either externally configured in a
2133 table or embedded in the operator with the inline notation (static).
- 2134 - *level* > 0 (static).
- 2135 - All the *MeasureComponents* of *ds_1* must be Numeric (static).

2136 Semantics

2137 It applies a hierarchical aggregation with name *hierarchyname* on an *IdentifierComponent*
2138 *comp* in a Dataset *ds_1*, aggregating all the *MeasureComponents* according to an aggregation
2139 function (algebraic sum or product).

2140 *hierarchyname* can be the identifier of an aggregation that is externally configured in a table,
2141 with an associated set of mappings, each with a sign and a level.

2142 Alternatively, a hierarchical aggregation can be expressed in an inline fashion, where
2143 *maps_from* constants are mapped into *maps_to* ones in the specific *level* and *sign*.

2144 For the given aggregation, for each level, all the mappings are considered and orderly applied
2145 with the following logic.

2146 For each value of the *IdentifierComponent* of all the records of the considered Dataset, if the
2147 value is present in `maps_from` for any mapping, it is turned into the respective `maps_to`
2148 value; if the value is not present in `maps_from` for any correspondence, the entire record is
2149 discarded if `isFilter` is true, the original value is preserved if `isFilter` is false.

2150 Aggregations are typically hierarchical, in the sense that they map many `maps_from` values
2151 into fewer `maps_to` values: often, multiple component Code Items collapse into the same
2152 compound Code Item.

2153 Therefore, at this stage, there may be multiple records having the same values for all the
2154 *IdentifierComponents*, as the differentiating ones have been aggregated into the same one.

2155 The records having the same value for all the *IdentifierComponents* are aggregated by
2156 algebraic sum (**sum**) or product (**prod**) of their *MeasureComponents*. If the aggregation
2157 function is omitted, sum is implied.

2158 Sum implies that the *MeasureComponents* have to be algebraically summed, considered with
2159 positive or negative sign depending on the Sign of the used mapping.

2160 Prod implies that the *MeasureComponents* have to be multiplied, considered with -1 exponent
2161 when the negative Sign is used in the mapping.

2162 Notice that the use of `prod` as aggregation function is meaningful only when the
2163 *IdentifierComponent* is a measure dimension.

2164 Hierarchies and measure dimensions

2165 As it is well known, an *IdentifierComponent* in a Dataset can play the role of *measure*
2166 *dimension*, meaning that the Dataset is indeed multi-measure, but represented as mono-
2167 measure with a further, measure-qualifying dimension.

2168 *IdentifierComponents* in `hierarchy` can also be a measure dimension, although in this case
2169 the aggregation inherently assumes a different meaning.

2170 In facts, Data Points with all coinciding *IdentifierComponents*, except for the measure
2171 dimension, are nothing but an expression of different measures for the same data point. An
2172 aggregation over the *measure dimension* is conceptually an operation involving the measures
2173 of the same Data Point (algebraic sum or multiplication).

2174 Hierarchies as transcodings

2175 The presented mapping method can be used intuitively to express a transcoding in a synthetic
2176 way. In this case, the involved Component value is mapped into another one, which is not
2177 hierarchically related with the first, but simply represents the same value expressed in
2178 another coding standard.

2179 Hierarchies as filters

2180 `isFilter` parameter in `hierarchy` allows choosing whether an input Data Point is to be kept
2181 in the output even if there are not mappings having the `maps_from` value corresponding to
2182 the *IdentifierComponent* of that Data Point.

2183 Indeed, this mechanism lends itself to the construction of reusable filters, independent of the
2184 specific Dataset and *IdentifierComponent* they are applied on.

2185 Such rules comprise a set of $(X \rightarrow X, Y \rightarrow, \dots, Z \rightarrow Z)$ correspondences preserving the values X, Y
 2186 and Z of the *IdentifierComponents* and filtering out the Data Points having a value for the
 2187 *IdentifierComponent* different from X, Y or Z.

2188

2189 Parameters

2190 ds_1 – the Dataset to be aggregated

2191 comp – the *IdentifierComponent* to aggregate upon

2192 hierarchyname – the name of the hierarchical aggregation of the information model, which
 2193 can be optionally replaced by an inline specification of the rule

2194 maps_from – an input value in an inline aggregation rule

2195 level – the level of hierarchy of a correspondence (which represents the order of
 2196 application) in an aggregation rule

2197 sign – the sign of the contribution of a maps_from value in an aggregation rule

2198 maps_to – an output value in an inline aggregation rule

2199 isFilter – if the aggregation must be interpreted as a filter (excluding non matching
 2200 records)

2201 Returns

2202 ds_2 – the aggregated Dataset

2203 Example 1

2204 The expression:

```
2205     Income_by_country_and_nace_ISO :=
2206     hierarchy("Income_by_country_and_nace_UN", GEO, un_to_ISO_aggr,
2207     false)
```

2208 or its equivalent inline form:

```
2209     Income_by_country_and_nace_ISO :=
2210     hierarchy("Income_by_country_and_nace_UN", GEO,
2211     (("ITA",1,+)) to "IT", (("ALB",1,+)) to AL, (("BEL",1,+)) to "BE")
2212     as un_to_ISO_aggr, false)
```

2213 Converts the values of the GEO Identifier Component from the United Nations 3-letter
 2214 standard into the ISO 2-letter one.

2215

HIERARCHY NAME	MAPS FROM	MAPS TO	START DATE	END DATE	SIGN	LEVEL	OUTPUT
un_to_ISO_aggr	ITA	IT	+	1	true
un_to_ISO_aggr	ALB	AL	+	1	true
un_to_ISO_aggr	BEL	BE	+	1	true

2216 Example 2

2217 The expression:

```
2218     Income_by_continent_and_nace :=  
2219     hierarchy("Income_by_state_and_nace", GEO, Continent_aggr, false)
```

2220 Takes as input the Dataset of the income, broken down by states and NACE and aggregates to
2221 continent level.

2222

HIERARCHY NAME	MAPS FROM	MAPS TO	START DATE	END DATE	SIGN	LEVEL	OUTPUT
Continent_aggr	Italy	Europe	+	1	true
Continent_aggr	France	Europe	+	1	true
Continent_aggr	Luxembourg	Europe	+	1	true
Continent_aggr	...	Europe			+	1	true
Continent_aggr	China	Asia			+	1	true
Continent_aggr	India	Asia			+	1	true
Continent_aggr	USA	America			+	1	true
Continent_aggr			+	1	true

2223

2224

2225

Income_by_state_and_nace

GEO	NACE	VALUE
Italy	IND	10
Italy	TECH	20
France	IND	31
France	TECH	50
Spain	IND	30
Spain	TECH	15
China	IND	250
China	TECH	250
India	IND	30
India	TECH	100
Luxembourg	IND	10
Luxembourg	TECH	12

2226

2227

Income_by_continent_and_nace

GEO	NACE	VALUE
Europe	IND	81
Europe	TECH	97
Asia	IND	280
Asia	TECH	350

2228

2229 Example 3

2230 The expression:

2231 `Income_by_world_and_nace := hierarchy("Income_by_state_and_nace",`
 2232 `GEO, World_aggr, false)`

2233 Takes as input the Dataset of the income, broken down by states and NACE and aggregates to
 2234 the world level. World_aggr is a 2 levels Rule. LEVEL 1 is only a temporary output and its
 2235 MAPS TO Code Items do not appear in the final result.

HIERARCHY NAME	MAPS FROM	MAPS TO	START DATE	END DATE	SIGN	LEVEL	OUTPUT
World_aggr	Italy	Europe	+	1	false
World_aggr	France	Europe	+	1	false
World_aggr	Luxembourg	Europe	+	1	false
World_aggr	...	Europe			+	1	false
World_aggr	...	Europe			+	1	false
World_aggr	China	Asia			+	1	false
World_aggr	India	Asia			+	1	false
World_aggr	USA	America			+	1	false
World_aggr	Asia	World			+	2	true
World_aggr	Europe	World			+	2	true
World_aggr	America	World			+	2	true
World_aggr	Oceania	World			+	2	true
World_aggr	Africa	World			+	2	true

2236

2237

Income_by_state_and_nace

GEO	NACE	VALUE
Italy	IND	10
Italy	TECH	20
France	IND	31

France	TECH	50
Spain	IND	30
Spain	TECH	15
China	IND	250
China	TECH	250
India	IND	30
India	TECH	100
Luxembourg	IND	10
Luxembourg	TECH	12

2238
2239

Income_world

GEO	NACE	VALUE
World	IND	361
World	TECH	447

2240

2241 Example 4

2242 The expression:

2243 `Income_world_and_nace_par := hierarchy("Income_by_state_and_nace",`
2244 `World_aggr_par, false)`

2245 Or its equivalent inline version:

2246 `Income_world_and_nace_par := aggrrule(`
2247 `Income_by_state_and_nace, GEO,`
2248 `((("Italy",1,+), ("France",1,+), ("Luxembourg",1,+)) to "Europe,`
2249 `((("China",1,+), ("India",1,+)) to "Asia",`
2250 `((("USA",1,+)) to "America,`
2251 `((("Asia",2,+), ("Europe",2,+), ("America",2,+),`
2252 `((("Oceania",2,+), ("Africa",2,+)) to "World")`
2253 `as World_aggr_par,`
2254 `false)`
2255

2256 Takes as input the Dataset of the income, broken down by states and NACE and aggregates to
2257 the World level. Differently from example 2, it preserves the first level in the output.

HIERARCHY NAME	MAPS FROM	MAPS TO	START DATE	END DATE	SIGN	LEVEL	OUTPUT
World_aggr_par	Italy	Europe	+	1	true
World_aggr_par	France	Europe	+	1	true
World_aggr_par	Luxembourg	Europe	+	1	true
World_aggr_par	...	Europe			+	1	true

World_aggr_par	...	Europe			+	1	true
World_aggr_par	China	Asia			+	1	true
World_aggr_par	India	Asia			+	1	true
World_aggr_par	USA	America			+	1	true
World_aggr_par	Asia	World			+	2	true
World_aggr_par	Europe	World			+	2	true
World_aggr_par	America	World			+	2	true
World_aggr_par	Oceania	World			+	2	true
World_aggr_par	Africa	World			+	2	true

2258

2259

Income_by_state_and_nace

GEO	NACE	VALUE
Italy	IND	10
Italy	TECH	20
France	IND	31
France	TECH	50
Spain	IND	30
Spain	TECH	15
China	IND	250
China	TECH	250
India	IND	30
India	TECH	100
Luxembourg	IND	10
Luxembourg	TECH	12

2260

2261

Income_world_and_nace_par

GEO	NACE	VALUE
World	IND	361
World	TECH	447
Europe	IND	81
Europe	TECH	97
Asia	IND	280
Asia	TECH	350

2262 Example 5

2263 The expression:

```
2264     Italian_income_by_nace:= hierarchy("Income_by_state_and_nace",
2265     Italy_filter, true)
```

2266 Takes as input the Dataset of the income, broken down by states and NACE and filters out all
 2267 the incomes that do not refer to Italy.

HIERARCHY NAME	MAPS FROM	MAPS TO	START DATE	END DATE	SIGN	LEVEL	OUTPUT
Italy_filter	Italy	Italy	+	1	true

2268

2269 Income_by_state_and_nace

GEO	NACE	VALUE
Italy	IND	10
Italy	TECH	20
France	IND	31
France	TECH	50
Spain	IND	30
Spain	TECH	15
China	IND	250
China	TECH	250
India	IND	30
India	TECH	100
Luxembourg	IND	10
Luxembourg	TECH	12

2270

2271 Italian_income_by_nace

GEO	NACE	VALUE
Italy	IND	10
Italy	TECH	20

2272

2273 Example 6

2274 The expression:

```
2275     income_prod_2:=     hierarchy("Income_by_state_and_nace_mm",     GEO,
2276     mult_rule, false, prod)
```

2277 Takes as input the Dataset of the income, broken down by states and NACE in a measure
 2278 dimension form and aggregates by calculating INC2 / INC1 into INC.

HIERARCHY NAME	MAPS FROM	MAPS TO	START DATE	END DATE	SIGN	LEVEL	OUTPUT
Mult_measure	INC1	INC	-	1	true
Mult_measure	INC2	INC	+	1	true

2279 **Income_by_state_and_nace**

GEO	NACE	MEASURE	VALUE
Italy	IND	INC1	10
Italy	IND	INC2	20
Italy	TECH	INC1	20
Italy	TECH	INC2	40
France	IND	INC1	31
France	IND	INC2	61
France	TECH	INC1	50
France	TECH	INC2	100
China	IND	INC1	250
China	IND	INC2	500
China	TECH	INC1	250
China	TECH	INC2	500
India	IND	INC1	30
India	IND	INC2	60
India	TECH	INC1	100
India	TECH	INC2	200

2280 **Income_prod2**

GEO	NACE	MEASURE	VALUE
Italy	IND	INC	2
Italy	TECH	INC	2
France	IND	INC	2
France	TECH	INC	2
China	IND	INC	2
China	TECH	INC	2
India	IND	INC	2
India	TECH	INC	2

2281 Validation operators

2282 VTL provides operators for **comparing** Datasets with one another and Datasets with
2283 Constants.

2284 For NULL values, the following holds for all relational operators: Since NULL does not
2285 represent a value, but instead is a marker indicating undefined, comparisons with NULL can
2286 never result in either True or False, but always in a third logical result, which is undefined and
2287 therefore best represented as NULL. This approach follows in principle the three-valued logic
2288 of Kleene and Łukasiewicz.

2289 Syntax

2290 Component<Boolean> c :=

2291 Component<T> c_1

2292 [$>$ | $<$ | \geq | \leq | $=$ | $<>$]

2293 Component<T> c_2

2294 Constraints

2295 T is not Boolean (static).

2296 Semantics

2297 Returns true if c_1 is greater-than, less-than, greater-or-equal, less-or-equal, equal, not equal
2298 c_2; otherwise false. For Numeric Component the meaning is straightforward, whereas for
2299 Strings, the comparison operators must be interpreted with respect to the lexicographical
2300 order.

2301 Examples

2302 If A = 5, B = 9, C = 5:

2303 D := A > B // D = false

2304 D := A < B // D = true

2305 D := A = B // D = false

2306 D := A \geq C // D = true

2307 D := A \leq C // D = true

2308 D := A = C // D = true

2309 D := A $<>$ C // D = false

2310 If A = "hello", B = "hi", C = "Hi":

2311 D := A > B // D = false

2312 D := C < B // D = false

2313 D := A = B // D = false

2314

2315 equal to =

2316 Syntax

```
2317 Dataset<?+, MeasureComponent<Boolean>+> ds_3 :=  
2318 [Dataset<T> | Constant<T>] ds_1  
2319 =  
2320 [Dataset<T> | Constant<T>] ds_2
```

2321 Constraints

- 2322 - at least one between *ds_1* and *ds_2* must be a Dataset (static).
- 2323 - if both *ds_1* and *ds_2* are Datasets they must have the same IdentifierComponents (in name and type) or, at least, the IdentifierComponents of *ds_1* may be a subset of the ones of *ds_2* (or viceversa) (static).
- 2326 - If both *ds_1* and *ds_2* are Datasets, they must have the same MeasureComponents, in name and number (static).
- 2328 - If *ds_1* (*ds_2*) is a Constant<T>, *ds_2* (*ds_1*) must have a single MeasureComponent<T> (static).

2330 Semantics

2331 It takes in input two Datasets *ds_1* and *ds_2* with the same IdentifierComponents, or at least
2332 the identifier components of one being a subset of the IdentifierComponents of the other, and
2333 the same MeasureComponents (in name and type). It returns a third Dataset *ds_3*, having the
2334 superset of the IdentifierComponents of *ds_1* and *ds_2*, some AttributeComponents (according
2335 to the attribute propagation rules) and, for each pair of homonym (and same type)
2336 MeasureComponents of *ds_1* and *ds_2* a MeasureComponent<Boolean>. For each pair of
2337 homonym MeasureComponents, the corresponding MeasureComponent<Boolean> in *ds_3* is
2338 named as the concatenation, separated by “_”, of their names with the suffix “CONDITION”,
2339 (e.g. if the compared MeasureComponents are named “VALUE”, the
2340 MeasureComponent<Boolean> will be named “VALUE_CONDITION”).

2341 For each pair of rows in *ds_1* and *ds_2* having the same values for all the common
2342 IdentifierComponents, the MeasureComponent<Boolean> in *ds_3* will contain the result of the
2343 comparison. Whenever a NULL is present in a comparison, NULL is returned as a result.

2344

2345 If *ds_1* (or *ds_2*) is a Constant, *ds_3* has a single MeasureComponent<Boolean>, named
2346 “CONDITION”, containing the result of the comparison of the Constant literal with the single
2347 MeasureComponent of all the rows.

2348 Parameters

2349 *ds_1* – a Dataset or a Constant

2350 *ds_2* – a Dataset or a Constant.

2351 Returns

2352 *ds_3* – a Dataset containing the result of the comparison.

2353

2354 Example

2355 In this example we check whether the values of a dataset measure are equal to a specific
2356 numeric value. The result dataset has the same classification structure as the initial dataset
2357 and it contains a single measure.

2358

overcrowding_rate_urbanization				
TIME	GEO	AGE	SEX	VALUE
2012	Belgium	Total	Total	NULL
2012	Greece	Total	Total	0.286
2012	Spain	Total	Total	0.064
2012	Malta	Total	Total	0.043
2012	Finland	Total	Total	0.08
2012	Switzerland	Total	Total	0.08

2359

2360 The expression:

2361 `Overcrowding_rate_urbanization = 0.08`

2362 will yield the following result dataset:

TIME	GEO	AGE	SEX	CONDITION
2012	Belgium	Total	Total	NULL
2012	Greece	Total	Total	false
2012	Spain	Total	Total	false
2012	Malta	Total	Total	false
2012	Finland	Total	Total	true
2012	Switzerland	Total	Total	true

2363

2364 not equal to <>

2365 Syntax

```

2366 Dataset<?+, MeasureComponent<Boolean>+> ds_3 :=
2367 [Dataset<T> | Constant<T>] ds_1
2368 <>
2369 [Dataset<T> | Constant<T>]

```

2370 Constraints

2371 - at least one between *ds_1* and *ds_2* must be a Dataset (static).

- 2372 - if both *ds_1* and *ds_2* are Datasets they must have the same IdentifierComponents (in
- 2373 name and type) or, at least, the IdentifierComponents of *ds_1* may be a subset of the
- 2374 ones of *ds_2* (or viceversa) (static).
- 2375 - If both *ds_1* and *ds_2* are Datasets, they must have the same MeasureComponents, in
- 2376 name and number (static).
- 2377 - If *ds_1* (*ds_2*) is a Constant<T>, *ds_2* (*ds_1*) must have a single MeasureComponent<T>
- 2378 (static).

2379 Semantics

2380 It takes in input two Datasets *ds_1* and *ds_2* with the same IdentifierComponents, or at least

2381 the identifier components of one being a subset of the IdentifierComponents of the other, and

2382 the same MeasureComponents (in name and type). It returns a third Dataset *ds_3*, having the

2383 superset of the IdentifierComponents of *ds_1* and *ds_2*, some AttributeComponents (according

2384 to the attribute propagation rules) and, for each pair of homonym (and same type)

2385 MeasureComponents of *ds_1* and *ds_2* a MeasureComponent<Boolean> For each pair of

2386 homonym MeasureComponents, the corresponding MeasureComponent<Boolean> in *ds_3* is

2387 named as the concatenation, separated by “_”, of their names with the suffix “CONDITION”,

2388 (e.g. if the compared MeasureComponents are named “VALUE”, the

2389 MeasureComponent<Boolean> will be named “VALUE_CONDITION”).

2390

2391 For each pair of rows in *ds_1* and *ds_2* having the same values for all the common

2392 IdentifierComponents, the MeasureComponent<Boolean> in *ds_3* will contain the result of the

2393 comparison not equal. Whenever a NULL is present in a comparison, NULL is returned as a

2394 result.

2395

2396 If *ds_1* (or *ds_2*) is a Constant, *ds_3* has a single MeasureComponent<Boolean>, named

2397 “CONDITION”, containing the result of the comparison of the Constant literal with the single

2398 MeasureComponent of all the rows.

2399 Parameters

2400 *ds_1* – a Dataset or a Constant

2401 *ds_2* – a Dataset or a Constant.

2402 Returns

2403 *ds_3* – a Dataset containing the result of the comparison.

2404 Example

Y_unemployment_2012				
GEO	SEX	UNIT	C_BIRTH	VALUE
Germany	Total	Percentage	Total	7.1
Greece	Total	Percentage	Total	NULL

2405

y_unemployment_2011				
GEO	SEX	UNIT	C_BIRTH	VALUE
Germany	Total	Percentage	Total	7.5
Greece	Total	Percentage	Total	3

2406

2407

2408

2409

The expression:

```
compare_ds := y_unemployment_2012 <> y_unemployment_2011
```

will yield the result:

GEO	SEX	UNIT	C_BIRTH	CONDITION_VALUE
Germany	Total	Percentage	Total	true
Greece	Total	Percentage	Total	NULL

2410

2411

2412

2413

If VALUE for Greece in the second operand had also been NULL, then the result would still be NULL for Greece. This reveals the shortcoming of our approach, namely that $(\text{NULL} \neq \text{NULL}) = (\text{NULL} == \text{NULL})$, which necessitates the introduction of an `isnull` operator in the language.

2414

greater than `>` `>=`

2415

Syntax

2416

2417

2418

2419

```
Dataset<?+, MeasureComponent<Boolean>+> ds_3 :=
[Dataset<T> | Constant<T>] ds_1
>{=}
[Dataset<T> | Constant<T>] ds_2
```

2420

Constraints

2421

2422

2423

2424

2425

2426

2427

2428

2429

- at least one between *ds_1* and *ds_2* must be a Dataset (static).
- if both *ds_1* and *ds_2* are Datasets they must have the same IdentifierComponents (in name and type) or, at least, the IdentifierComponents of *ds_1* may be a subset of the ones of *ds_2* (or viceversa) (static).
- If both *ds_1* and *ds_2* are Datasets, they must have the same MeasureComponents, in name and number (static).
- If *ds_1* (*ds_2*) is a Constant<T>, *ds_2* (*ds_1*) must have a single MeasureComponent<T>.
- T must not be Boolean (static).

2430

Semantics

2431

2432

2433

2434

2435

It takes in input two Datasets *ds_1* and *ds_2* with the same IdentifierComponents, or at least the identifier components of one being a subset of the IdentifierComponents of the other, and the same MeasureComponents (in name and type). It returns a third Dataset *ds_3*, having the superset of the IdentifierComponents of *ds_1* and *ds_2*, some AttributeComponents (according to the attribute propagation rules) and, for each pair of homonym (and same type)

2436 MeasureComponents of *ds_1* and *ds_2* a MeasureComponent<Boolean> For each pair of
 2437 homonym MeasureComponents, the corresponding MeasureComponent<Boolean> in *ds_3* is
 2438 named as the concatenation, separated by “_”, of their names with the suffix “CONDITION”,
 2439 (e.g. if the compared MeasureComponents are named “VALUE”, the
 2440 MeasureComponent<Boolean> will be named “VALUE_CONDITION”).

2441 For each pair of rows in *ds_1* and *ds_2* having the same values for all the common
 2442 IdentifierComponents, the MeasureComponent<Boolean> in *ds_3* will contain the result of the
 2443 comparison greather (or equal) than. For Numeric MeasureComponents, the meaning is
 2444 straightforward, whereas for Strings, the comparison operators must be interpreted with
 2445 respect to the lexicographical order.

2446 If *ds_1* (or *ds_2*) is a Constant, *ds_3* has a single MeasureComponent<Boolean>, named
 2447 “CONDITION”, containing the result of the comparison of the Constant literal with the single
 2448 MeasureComponent of all the rows.

2449 Parameters

2450 *ds_1* – a Dataset or a Constant

2451 *ds_2* – a Dataset or a Constant.

2452 Returns

2453 *ds_3* – a Dataset containing the result of the comparison.

2454 Example 1

2455 In this example, we check whether the values of a dataset measure are greater than a Constant
 2456 value. The result dataset has the same classification structure as the initial dataset and it
 2457 contains a single measure.

foreign_languages_known					
N_LANG	GEO	TIME	AGE	UNIT	VALUE
2	Germany	2011	Total	Percentage	NULL
2	Greece	2011	Total	Percentage	12.2
2	Finland	2011	Total	Percentage	29.5

2458

2459 The expression:

2460 `compare_ds := foreign_languages_known > 20`

2461 will yield the following result dataset:

compare_ds					
N_LANG	GEO	TIME	AGE	UNIT	CONDITION
2	Germany	2011	Total	Percentage	NULL
2	Greece	2011	Total	Percentage	false
2	Finland	2011	Total	Percentage	true

2462 Example 2

2463 In the following example we check whether the values of a dataset measure are greater than
 2464 the values of another dataset measure. We use data about the youth unemployment rate of
 2465 two consecutive years.

2466

y_unemployment_2012				
GEO	SEX	UNIT	C_BIRTH	VALUE
Germany	Total	Percentage	Total	7.1
Greece	Total	Percentage	Total	42.5

2467

2468

y_unemployment_2011				
GEO	SEX	UNIT	C_BIRTH	VALUE
Germany	Total	Percentage	Total	7.5
Greece	Total	Percentage	Total	33.7

2469

2470 The expression

2471 `compare_ds := y_unemployment_2012 > y_unemployment_2011`

2472

2473 will yield the result:

compare_ds				
GEO	SEX	UNIT	C_BIRTH	CONDITION_VALUE
Germany	Total	Percentage	Total	false
Greece	Total	Percentage	Total	true

2474

2475 If the VALUE column for Germany in the y_unemployment_2012 dataset had a NULL
 2476 value the result would be:

2477

GEO	SEX	UNIT	C_BIRTH	CONDITION_VALUE
Germany	Total	Percentage	Total	NULL
Greece	Total	Percentage	Total	true

2478

2479 less than < <=

2480 Syntax

```
2481 Dataset<?+, MeasureComponent<Boolean>+> ds_3 :=  
2482 [Dataset<T> | Constant<T>] ds_1  
2483 <{=}  
2484 [Dataset<T> | Constant<T>] ds_2
```

2485 Constraints

- 2486 - at least one between *ds_1* and *ds_2* must be a Dataset (static).
- 2487 - if both *ds_1* and *ds_2* are Datasets they must have the same IdentifierComponents (in name and type) or, at least, the IdentifierComponents of *ds_1* may be a subset of the ones of *ds_2* (or viceversa) (static).
- 2489 - If both *ds_1* and *ds_2* are Datasets, they must have the same MeasureComponents, in name and number (static).
- 2492 - If *ds_1* (*ds_2*) is a Constant<T>, *ds_2* (*ds_1*) must have a single MeasureComponent<T> (static).

2494 Semantics

2495 It takes in input two Datasets *ds_1* and *ds_2* with the same IdentifierComponents, or at least
2496 the identifier components of one being a subset of the IdentifierComponents of the other, and
2497 the same MeasureComponents (in name and type). It returns a third Dataset *ds_3*, having the
2498 superset of the IdentifierComponents of *ds_1* and *ds_2*, some AttributeComponents (according
2499 to the attribute propagation rules) and, for each pair of homonym (and same type)
2500 MeasureComponents of *ds_1* and *ds_2* a MeasureComponent<Boolean>. For each pair of
2501 homonym MeasureComponents, the corresponding MeasureComponent<Boolean> in *ds_3* is
2502 named as the concatenation, separated by “_”, of their names with the suffix “CONDITION”,
2503 (e.g. if the compared MeasureComponents are named “VALUE”, the
2504 MeasureComponent<Boolean> will be named “VALUE_CONDITION”).

2505 For each pair of rows in *ds_1* and *ds_2* having the same values for all the common
2506 IdentifierComponents, the MeasureComponent<Boolean> in *ds_3* will contain the result of the
2507 comparison less (or equal) than. Whenever a NULL is present in a comparison, NULL is
2508 returned as a result.

2509 If *ds_1* (or *ds_2*) is a Constant, *ds_3* has a single MeasureComponent<Boolean>, named
2510 “CONDITION”, containing the result of the comparison of the Constant literal with the single
2511 MeasureComponent of all the rows.

2512 Parameters

2513 *ds_1* – a Dataset or a Constant

2514 *ds_2* – a Dataset or a Constant.

2515 Returns

- 2516 - *ds_3* – a Dataset containing the result of the comparison.

2517 Example

2518 In this example, we check whether the values of a Dataset are smaller than a specific Constant
2519 numeric value. The result dataset has the same classification structure as the initial dataset
2520 and it contains a single measure.

2521

total_population				
TIME	GEO	AGE	SEX	VALUE
2012	Belgium	Total	Total	11094850
2012	Greece	Total	Total	11123034
2012	Spain	Total	Total	46818219
2012	Malta	Total	Total	NULL
2012	Finland	Total	Total	5401267
2012	Switzerland	Total	Total	7954662

2522

2523 The expression:

2524 `compare_ds := total_population < 15000000`

2525 will yield the following result dataset:

compare_ds				
TIME	GEO	AGE	SEX	CONDITION
2012	Belgium	Total	Total	true
2012	Greece	Total	Total	true
2012	Spain	Total	Total	false
2012	Malta	Total	Total	NULL
2012	Finland	Total	Total	true
2012	Switzerland	Total	Total	true

2526

2527 **in, not in**

2528 **Component version**

2529 Syntax

2530 `Component<T> c_1 {not}`

2531 **in**

2532 `{Set|List}<T> c_2`

2533 Constraints

2534 None

2535 Semantics

2536 Returns true if *c_1* is (not) in (set or list) *c_2*.

2537 **Dataset version**

2538 Syntax

```

2539     Dataset<?+, MeasureComponent<Boolean>+> ds_2 := Dataset<?+,
2540     MeasureComponent<T>+> ds_1
2541     {not} in
2542     {CollectionSet|CollectionList}<T> s

```

2543 Constraints

- all the MeasureComponents of *ds_1* must have the same type T (which is also the type of the Set or List) (static).

2546 Semantics

2547 Checks whether the MeasureComponents of the Dataset *ds_1* take values in the Set or List *s*. It
 2548 returns a second Dataset *ds_2*, with all the IdentifierComponents in *ds_1*, all the
 2549 AttributeComponents in *ds_1* (according to the attribute propagation rule) and a
 2550 MeasureComponent<Boolean> for each MeasureComponent<T> of *ds_1*, named
 2551 "CONDITION_" concatenated to the name of the original MeasureComponent. For each row in
 2552 *ds_1* there is a row in *ds_2* where the MeasureComponents states if the values of *ds_1* are (not)
 2553 in *s*.

2554 Parameters

2555 *ds_1* – a Dataset

2556 *s* – a CollectionSet or CollectionList of values

2557 Returns

2558 *ds_2* – a Dataset containing the result of the comparison.

2559 Example

2560 In this example we apply the {not} in operator on the GEO dimension of the total_population
 2561 Dataset.

2562

total_population				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	11094850
2012	Greece	Total	Total	11123034

2012	Spain	Total	Total	46818219
2012	Malta	Total	Total	417546
2012	Finland	Total	Total	5401267
2012	NULL	Total	Total	7954662

2563

2564

The result yielded by the expression

2565

```
ds_1 := total_population in (11094850, 46818219, 222, 111)
```

2566

will have a single measure column that will take value *true* for records for which the value in the POPULATION ComponentMeasure is contained in the set, *false* otherwise.

2567

TIME	GEO	AGE	SEX	CONDITION_POPULATION
2012	Belgium	Total	Total	true
2012	Greece	Total	Total	false
2012	Spain	Total	Total	true
2012	Malta	Total	Total	false
2012	Finland	Total	Total	false
2012	NULL	Total	Total	false

2568 **between**

2569 **Component version**

2570 Syntax

2571 Component<Boolean> c :=

2572 Component<T> c_1

2573 **between**

2574 Component<T> c_2

2575 **and**

2576 Component<T> c_3

2577 Constraints

2578 None

2579 Semantics

2580 Returns True if c_1 is greater or equal than c_2 and less or equal than c_3; otherwise false.

2581 **Dataset version**

2582 Syntax

2583 Dataset<?+, MeasureComponent<Boolean>+> ds_4 :=

2584 [Dataset<T> | Constant<T>] *ds_1*
2585 {**not**} **between**
2586 [Dataset<T> | Constant<T>] *ds_2*
2587 **and**
2588 [Dataset<T> | Constant<T>] *ds_3*

2589 Constraints

- 2590 - at least one between *ds_1*, *ds_2* and *ds_3* must be a Dataset (static).
- 2591 - if two (or three) among *ds_1*, *ds_2* and *ds_3* are Datasets, for every pair of Datasets, it
2592 must hold that either they have the same IdentifierComponents or the ones of the
2593 former is a subset of the ones of the latter (static).
- 2594 - If two (or three) among *ds_1*, *ds_2* and *ds_3* are Datasets, they must have the same
2595 MeasureComponents, in name and number (static).
- 2596 - If at least one among *ds_1*, *ds_2* or *ds_3* is a Constant, the others must have a single
2597 MeasureComponent<T> (static).

2598 Semantics

2599 It takes in input three Datasets *ds_1*, *ds_2* and *ds_3* with the described constraints and returns
2600 a fourth Dataset *ds_4* with the most comprehensive set of IdentifierComponents out of the
2601 ones of *ds_1*, *ds_2* and *ds_3*, all the AttributeComponents (according to attribute propagation
2602 rules) and, for each triple of homonym (and same type) MeasureComponent, a
2603 MeasureComponent<Boolean>.

2604 For each triple of homonym MeasureComponents, the corresponding
2605 MeasureComponent<Boolean> in *ds_3* is named as the concatenation, separated by “_”, of
2606 their names (e.g. if the compared MeasureComponents are named “VALUE”, the
2607 MeasureComponent<Boolean> will be named “VALUE_CONDITION”).

2608 For each triple of rows in *ds_1*, *ds_2* and *ds_3* having the same values for all the common
2609 IdentifierComponents, the MeasureComponent<Boolean> in *ds_3* will contain the result of the
2610 range comparison: *true* (*false*) if the MeasureComponent value of *ds_1* is (not) between the
2611 values of *ds_2* and *ds_3*.

2612 If at least one among *ds_1*, *ds_2* or *ds_3* is a Constant, *ds_4* has a single
2613 MeasureComponent<Boolean>, named “CONDITION”, containing the result of the comparison
2614 of the Constant literal(s) with the single MeasureComponent of the Dataset(s).

2615 Parameters

2616 *ds_1* – a Dataset or a Constant

2617 *ds_2* – a Dataset or a Constant

2618 *ds_3* – a Dataset or a Constant.

2619 Returns

2620 *ds_4* – a Dataset containing the result of the range comparison.

2621 Example 1

2622 In the following example, we filter the years with unemployment rate between 7.5 and 8.

2623

unemployment_rate		
TIME	GEO	UNEMPLOYMENT_RATE
2013	Finland	8.2
2012	Finland	7.7
2011	Finland	7.8
2010	Finland	8.4
2009	Finland	NULL

2624

2625

2626

2627

The expression

`comparison_ds := unemployment_rate between 7.5 and 8.0`

will yield:

comparison_ds		
TIME	GEO	UNEMPLOYMENT_RATE
2013	Finland	false
2012	Finland	true
2011	Finland	true
2010	Finland	false
2009	Finland	NULL

2628

2629

Example 2

2630

2631

2632

In the following example we use a Dataset showing the overcrowding rate in densely populated areas.

overcrowding_rate_urbanization_2011	
GEO	VALUE
Belgium	NULL
Greece	0.276
Finland	0.093
Switzerland	0.08
United Kingdom	0.089
France	0.125

2633

overcrowding_rate_urbanization_2010			
GEO	AGE	SEX	VALUE
Belgium	Total	Total	0.06
Greece	Total	Total	0.281
Spain	Total	Total	0.06
Malta	Total	Total	0.041
Switzerland	Total	Total	NULL

2634

overcrowding_rate_urbanization_2012				
TIME	GEO	AGE	SEX	VALUE
2012	Belgium	Total	Total	0.023
2012	Greece	Total	Total	0.286
2012	Spain	Total	Total	0.064
2012	Malta	Total	Total	0.043
2012	Finland	Total	Total	0.08
2012	Switzerland	Total	Total	0.08

2635

2636

The expression:

2637

`comparison_ds := overcrowding_rate_urbanization_2011 between`

2638

`Overcrowding_rate_urbanization_2010 and`

2639

`Overcrowding_rate_urbanization_2012`

2640

2641

2642

will yield the result:

TIME	GEO	AGE	SEX	CONDITION_VALUE
2012	Belgium	Total	Total	NULL
2012	Greece	Total	Total	false
2012	Switzerland	Total	Total	NULL

2643

2644 **isnull**

2645 **Component version**

2646 Syntax

```
2647     Component<Boolean> c :=  
2648     isnull  
2649     ({component=}Component<T> c_1)
```

2650 Constraints

2651 None

2652 Semantics

2653 Returns *true* if *c_1* is null, *false* otherwise.

2654 Examples

2655 Let us assume component C from a dataset.

2656 If C is null:

```
2657     A := isnull(C) // A = true
```

2658 If C is not null:

```
2659     A := isnull(C) // A = false
```

2660 **Dataset version**

2661 Syntax

```
2662     Dataset<?+, MeasureComponent<Boolean>+> ds_2 :=  
2663     isnull  
2664     ({dataset=}Dataset<?> ds_1)
```

2665 Constraints

2666 None

2667 Semantics

2668 Checks if the rows of *ds_1* have NULL MeasureComponents. Returns a Dataset *ds_1* with the
2669 same IdentifierComponents as *ds_1*, the AttributeComponents (according to the attribute
2670 propagation rule) and a MeasureComponent<Boolean>, named “CONDITION_” concatenated
2671 to the name of the original MeasureComponent, for each MeasureComponent of *ds_1*. For each
2672 row in *ds_1* there is a row in *ds_2* with *true* or *false* as MeasureComponent stating if *ds_1*
2673 MeasureComponents are NULL or not NULL.

2674 Parameters

2675 - *ds_1* – the Dataset to be checked

2676 Returns

2677 - `ds_2` – a Dataset with the result of the check
2678

2679 Example

2680

population				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	11094850
2012	Greece	Total	Total	11123034
2012	Spain	Total	Total	NULL
2012	Malta	Total	Total	417546
2012	Finland	Total	Total	5401267
2012	NULL	Total	Total	NULL

2681 The expression

2682 `ds_1 := isnull(population)`

2683 applied against the previous Dataset, returns:

ds_1				
TIME	GEO	AGE	SEX	CONDITION
2012	Belgium	Total	Total	false
2012	Greece	Total	Total	false
2012	Spain	Total	Total	true
2012	Malta	Total	Total	false
2012	Finland	Total	Total	false
2012	NULL	Total	Total	true

2684 exists_in, not_exists_in/in_all

2685 Syntax

2686 `Dataset<?, MeasureComponent<Boolean>> ds_3 :=`

2687 `Dataset<?> ds_1`

2688 `[exists_in|exists_in_all|not_exists_in|not_exists_in_all]`

2689 `Dataset<?> ds_2`

2690 Constraints

2691 - *ds_1* and *ds_2* must have the same IdentifierComponents in name and type or, at least,
2692 the IdentifierComponents of *ds_1* (*ds_2*) must be a subset of the ones of *ds_2* (*ds_1*)
2693 (static).

2694 Semantics

2695 Checks that rows in the first argument have matching rows in the second argument.

2696 Takes in input two Datasets *ds_1* and *ds_2* with the same IdentifierComponents or, at least, the
2697 IdentifierComponents of *ds_1* (*ds_2*) being a subset of the ones of *ds_2* (*ds_1*), and returns a
2698 third Dataset *ds_3* with all the IdentifierComponents of the two and one
2699 MeasureComponent<Boolean> named "CONDITION".

2700 The MeasureComponent value in each row in *ds_3* indicates whether a row with matching key
2701 (not) exists in the second argument for the corresponding row of the first argument.

2702 If **all** versions are used, both the *true* and the *false* rows are kept in the result. Otherwise, only
2703 the *true* rows are kept.

2704 Parameters

2705 *ds_1* – the operand whose IdentifierComponents have to be looked up in *ds_2*

2706 *ds_2* – the reference Dataset in which *ds_1* IdentifierComponents are looked up

2707 Returns

2708 *ds_3* – The output Dataset with the result of the comparison

2709 Example 1

2710 Let us assume the following datasets:

2711 **population**

TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	11094850
2012	Greece	Total	Total	11123034
2012	Spain	Total	Total	46818219
2012	Malta	Total	Total	417546
2012	Finland	Total	Total	5401267
2012	Switzerland	Total	Total	7954662

2712 **urbanization_rate**

TIME	GEO	AGE	SEX	RATE
2012	Belgium	Total	Total	0.023
2012	NULL	Total	Total	0.286
2012	Spain	Total	Total	0.064

2012	Malta	Total	M	0.043
2012	Finland	NULL	Total	NULL
2012	Switzerland	Total	Total	0.08

2713

2714

The expression

2715

`ds_check := population exists_in_all urbanization_rate`

2716

will result into the following dataset:

TIME	GEO	AGE	SEX	CONDITION
2012	Belgium	Total	Total	true
2012	Greece	Total	Total	false
2012	Spain	Total	Total	true
2012	Malta	Total	Total	false
2012	Finland	Total	Total	false
2012	Switzerland	Total	Total	true

2717

2718 In the following examples, Ki are Identifier Components, Mi are Measure Components

2719 Example 2

2720

`R := C1 exists_in C2`

2721

C1		
K1	K2	M1
1	A	100
2	B	200
3	C	700
4	A	550
5	D	120

2727

2728

C2		
K1	K2	M1
1	A	100
2	B	200
5	D	700

2729

2730

2731

2732

2733

2734

2735

2736

2737

2738

2739

2740

R		
K1	K2	CONDITION
1	A	True
2	B	True
5	D	True

2741 Example 3

2742 R := C1 exists_in_all C2

2743

2744

2745

2746

2747

2748

2749

2750

C1		
K1	K2	M1
1	A	100
2	B	200
3	C	700
4	A	550
5	D	120

C2		
K1	K2	M1
1	A	100
2	B	200
5	D	700

2751 Example 4

2752 R := C1 does not_exist_in C2

2753

2754

2755

2756

2757

2758

2759

2760

C1		
K1	K2	M1
1	A	100
2	B	200
3	C	700
4	A	550
5	D	120

R		
K1	K2	CONDITION
1	A	True
2	B	True
3	C	False
4	A	False
5	D	True

2761

2762

2763

2764

C2		
K1	K2	M1
1	A	100
2	B	200
5	D	700

2765
2766
2767
2768
2769
2770
2771
2772

R		
K1	K2	CONDITION
3	C	True
4	A	True

2773 Example 5

2774 R := C1 not_exists_in_all C2-
2775

2776
2777
2778
2779
2780
2781
2782

C1		
K1	K2	M1
1	A	100
2	B	200
3	C	700
4	A	550
5	D	120

C2		
K1	K2	M1
1	A	100
2	B	200
5	D	700

2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793

R		
K1	K2	CONDITION
1	A	False
2	B	False
3	C	True
4	A	True
5	D	False

2794 Example 6

2795 R := C1 not_exists_in_all C2
2796

C1		
K1	K2	M1
1	A	100
2	B	200

3	C	700
4	A	550
5	D	120

2797

C2			
K1	K2	K3	M1
1	A	X	100
2	B	Y	200
5	D	Z	700
5	D	K	1500

2798

R		
K1	K2	CONDITION
1	A	False
2	B	False
3	C	True
4	A	True
5	D	False

2799

2800 check

2801 Syntax

```

2802 Dataset<?+, MeasureComponent<Boolean>> ds_2 :=
2803 check (
2804   {dataset=}Dataset<?+, MeasureComponent<Boolean>> ds_1
2805     {, imbalance(Dataset<Numeric> imbalance)}
2806     {, erlevel(Dataset<Numeric> errorlevel)}
2807     {, errorCode(Constant<String> errorCode)}
2808     {, threshold(Constant<Numeric> threshold)}
2809     {, all})

```

2810 Constraints

2811 - *ds_1* must have exactly one MeasureComponent<Boolean> (static).

2812 Semantics

2813 It is the validation operator: it allows to enrich and control a comparison condition, expressed
 2814 as a Boolean Dataset, with supplementary properties, essential for a validation.

2815 *Imbalance*, *errorlevel* and *errorCode* are extra indicators for the validation, while *threshold*
2816 allows to relax the comparison adding a tolerance.

2817 Specifically, it takes as input a Dataset *ds_1* having exactly one MeasureComponent<Boolean>
2818 and returns a Dataset with the same IdentifierComponents, MeasureComponents and
2819 AttributeComponents (according to attribute propagation rules) with at most three
2820 supplementary MeasureComponents whose values are calculated row-wise:

- 2821 - *imbalance* is a Dataset<Numeric> expression typically over the Datasets possibly used
2822 in calculating *ds_1*, calculating in its single MeasureComponent<Numeric> the
2823 imbalance between an expected value for each row in *ds_1*. The MeasureComponent is
2824 named IMBALANCE. This value is returned as 0 if the row has *true* in the
2825 MeasureComponent, otherwise it is calculated.
- 2826 - *errorlevel* is Dataset<Numeric> expression typically over the Datasets possibly used in
2827 calculating *ds_1*, calculating an in its single MeasureComponent<Numeric> an error
2828 level for each row in *ds_1*. This value is returned as 0 if the row has *true* in the
2829 MeasureComponent, otherwise it is calculated. The errorlevel is typically used by the
2830 following statements to behave accordingly. The MeasureComponent is named
2831 ERRORLEVEL.
- 2832 - *errorCode* the error code of a validation error to be returned when the row has *false* in
2833 the MeasureComponent. In the other cases it is NULL. The MeasureComponent storing
2834 this value is named ERRORCODE.

2835
2836 The **all** flag allows to specify that both *true* and *false* rows have to be kept in the output. If it is
2837 not present, only *true* rows are kept.

2838 Parameters

2839 *ds_1* – the Dataset with the check

2840 *imbalance* – the Dataset<Numeric> calculating the imbalance

2841 *errorlevel* – the Dataset<Numeric> calculating an error level for the check

2842 *errorCode* – an error code to be used when the check is false.

2843 *threshold* - a tolerance threshold to be used when reporting the outcome of the validation.
2844 This parameter allows relaxing the check enforced in *ds_1*. For every row where the
2845 MeasureComponent is false (hence the validation would fail), if *imbalance* < *threshold*, then
2846 such MeasureComponent is turned into true and the validation succeeds.

2847

2848 Returns

2849 *ds_2* – the output Dataset, which is a copy of the input one plus the IMBALANCE,
2850 ERRORLEVEL, ERRORMESSAGE Components.

2851

2852

2853 Example 1

2854 `R := check(C1 > C2, imbalance (abs(C1-C2)) , erlevel (8))`

2855

C1		
K1	K2	M1
1	A	100
2	B	200

2856

2857

2858

2859

2860

C2		
K1	K2	M1
1	A	50
2	B	260

2861

2862

2863

2864

R				
K1	K2	CONDITION	IMBALANCE	ERRORLEVEL
1	A	true	0	0

2865 Example 2

2866 `R := check(C1 > C2, imbalance (abs(C1-C2)) , erlevel (8) , all)`

2867

C1		
K1	K2	M1
1	A	100
2	B	200

2868

2869

2870

C2		
K1	K2	M1
1	A	50
2	B	260

2871

2872

2873

2874

R				
K1	K2	CONDITION	IMBALANCE	ERRORLEVEL
1	A	true	0	0
2	B	false	60	8

2875 Example 3

2876 `R := check(C1 > =C2 * 0.1, imbalance (C1), erlevel (C1 * 0.1), all)`

2877

2878

2879

2880

2881

C1		
K1	K2	M1
1	A	100
2	B	200

2882

R					
K1	K2	K3	CONDITION	IMBALANCE	ERRORLEVEL
1	A	X	true	0	0
2	B	Y	true	0	0
2	B	Z	true	0	0

C2			
K1	K2	K3	M1
1	A	X	1000
2	B	Y	200
2	B	Z	350

2883

2884 Example 4

```
2885 R := check(C1 > =C2 * 0.1, imbalance (C1), erlevel (C1 * 0.1),
2886 threshold(210), all)
```

2887

2888

2889

2890

2891

C1		
K1	K2	M1
1	A	100
2	B	200

2892

2893

2894

2895

2896

2897

2898

R					
K1	K2	K3	CONDITION	IMBALANCE	ERRORLEVEL
1	A	X	true	0	0
2	B	Y	true	200	20
2	B	Z	true	0	0

2899

2900 match_characters

2901 Syntax

```
2902 Dataset<?+, MeasureComponent<Boolean>+> ds_2 :=  
2903 match_characters  
2904 ({dataset=}Dataset<?+, MeasureComponent<String>> ds_1,  
2905 {pattern=}Constant<String> pattern {, all})
```

2906 Constraints

- 2907 - *ds_1* must have only one MeasureComponents<String> (static).
- 2908 - *pattern* must be a regular expression according to POSIX extended standard
2909 (http://pubs.opengroup.org/onlinepubs/009696899/basedefs/xbd_chap09.html)
2910 (static).

2912 Semantics

2913 It takes in input a Dataset *ds_1* and a regular expression pattern *pattern* and returns a second
2914 Dataset *ds_2*, having the same IdentifierComponents, AttributeComponents (according to
2915 attribute propagation rule) and a MeasureComponent<Boolean> for each
2916 MeasureComponent<String> in *ds_1*.

2917 These MeasureComponents<Boolean> are named with the name of the MeasureComponent in
2918 *ds_1* concatenated with the suffix “_CONDITION”.

2919

2920 The rows of *ds_1* are copied into *ds_2*; the MeasureComponent<Boolean> will have *true* if the
2921 respective in *ds_1* matches with the *pattern*, *false* otherwise.

2922 The **all** flag allows to specify that both *true* and *false* rows have to be kept in the output. If it is
2923 not present, only *true* rows are kept.

2924 Parameters

2925 *ds_1* – the Dataset to be checked against the *pattern*

2926 *pattern* – a regular expression pattern according to POSIX extended standard
2927 (http://pubs.opengroup.org/onlinepubs/009696899/basedefs/xbd_chap09.html)

2928 Returns

2929 *ds_2* – a Dataset with the checked condition

2930

2931 Example

2932 In this validation rule example we check if all values for the Year IdentifierComponent are
2933 composed of numeric digits. The result dataset has the same Components as the argument
2934 dataset and it contains a single MeasureComponent, holding the check.

2935

2936 **population**

TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	11094850
2012A	Greece	Total	Total	11123034
2012	Spain	Total	Total	46818219
2012	Malta	Total	Total	417546
2012	Finland	Total	Total	5401267
2012C	Switzerland	Total	Total	7954662

2937 The expression:

2938 `ds_r := match_characters(population, TIME, "[123456789,]", all))`

2939 will yield the result:

TIME	GEO	AGE	SEX	POPULATION_CONDITION
2012	Belgium	Total	Total	true
2012A	Greece	Total	Total	false
2012	Spain	Total	Total	true
2012	Malta	Total	Total	true
2012	Finland	Total	Total	true
2012C	Switzerland	Total	Total	false

2940 match_values

2941 Syntax

```

2942 Dataset<?+, MeasureComponent<Boolean>+> ds_2 :=
2943 match_values
2944 ({dataset=} Dataset<?+, MeasureComponent<String>> ds_1,
2945 {values=}
2946 [ValueDomainSubset<?> | ConstantSet< <String>] vds {, all})
2947
2948
```

2949 Constraints

2950 - `ds_1` must have only one *MeasureComponent*<String> (static).

2951 Semantics

2952 It takes in input a *Dataset* `ds_1` and a Value Domain Subset (or a collection of allowed value
2953 codes in the form of a Set of Strings) `vds` and returns a second *Dataset* `ds_2`, having the same
2954 *IdentifierComponents*, *AttributeComponents* (according to attribute propagation rule) and a
2955 *MeasureComponent*<Boolean> for each *MeasureComponent*<String> in `ds_1`. These
2956 *MeasureComponents*<Boolean> are named "CONDITION_" concatenated to the name of the
2957 *MeasureComponent* in `ds_1`.

2958 The rows of `ds_1` are copied into `ds_1`; the *MeasureComponent*<Boolean> will have `true` if
2959 the respective in `ds_1` has a value that is contained within the Value Domain Subset,
2960 otherwise `false`.

2961 The `all` flag allows specifying that both `true` and `false` rows have to be kept in the output. If
2962 it is not present, only `true` rows are kept.

2963 Parameters

2964 `ds_1` – the Dataset to be checked against the *pattern*

2965 `vds` – the Value Domain Subset to be looked up or a collection of allowed codes in the form of
2966 a Set of Strings

2967 Returns

2968 `ds_2` – a Dataset with the checked condition

2969

2970 Example

2971 In this validation rule example we check if values in the Dataset *MeasureComponent* “SEX” are
2972 taken from a specific Value Domain Subset.

2973

2974

2975

unemployment

TIME	GEO	SEX	CITIZEN	AGE	WSTATUS
2008	IT	T	TOTAL	Y_LT15	UNE
2008	IT	T	TOTAL	Y15-19	UNE
2008	IT	T	TOTAL	Y20-64	UNE
2008	IT	T	TOTAL	Y_GE65	UNE
2008	IT	M	TOTAL	Y15-19	UNE
2008	IT	M	TOTAL	Y20-64	UNE
2008	IT	M	TOTAL	Y_GE65	UNE
2008	IT	F	TOTAL	Y15-19	UNE
2008	IT	F	TOTAL	Y20-64	UNE
2008	IT	F	TOTAL	Y_GE65	UNE
2008	NULL	F	TOTAL	Y_GE65	UNE

2976

2977 The expression:

2978 `r:= match_values (unemployment#SEX, ("M","F"), all)`

2979 will result into the following dataset:

TIME	GEO	SEX	CITIZEN	AGE	WSTATUS	CONDITION_SEX
2008	IT	T	TOTAL	Y_LT15	UNE	false
2008	IT	T	TOTAL	Y15-19	UNE	false
2008	IT	T	TOTAL	Y20-64	UNE	false
2008	IT	T	TOTAL	Y_GE65	UNE	false
2008	IT	M	TOTAL	Y15-19	UNE	true
2008	IT	M	TOTAL	Y20-64	UNE	true
2008	IT	M	TOTAL	Y_GE65	UNE	true
2008	IT	F	TOTAL	Y15-19	UNE	true
2008	IT	F	TOTAL	Y20-64	UNE	true
2008	IT	F	TOTAL	Y_GE65	UNE	true

2980

2981 Conditional operator

2982 VTL allows conditional evaluation via the if-then-else operator.

2983 if-then-else

2984 Component version

2985 Syntax

```
2986 Component<T> c :=  
2987 if Component<Boolean> cond_1  
2988 then Component<T> c_1  
2989 {elseif Component<Boolean> cond_2  
2990 then Component<T> c_2}*  
2991 else Component<T> c_3
```

2992 Constraints

2993 None

2994 Semantics

2995 If *cond_1* evaluates to true returns *c_1*; alternatively, if *cond_2* evaluates to true returns *c_2*.
2996 Otherwise returns *c_3*.

2997 Examples

2998 Expressions evaluating to Component types are typically used to calculate Measure
2999 Components or evaluate filters.

3000 Some examples follow:

```
3001 K1 + K2 < K3  
3002 K1 - K2 > 5.5  
3003 K2 + round(K2, 3)  
3004 K1 > 3 and k1 < 5  
3005 if k1>4 then K2 else K3 + 3  
3006 K1 in (1,2,3,4) and K3 not in ('a','b','c')
```

3007 Dataset version

3008 Syntax

```
3009 Dataset<T> ds_4 :=  
3010 if Dataset<?+,MeasureComponent<Boolean>> ds_cond_1  
3011 then Dataset<T> ds_1  
3012 {elseif Dataset<?+,MeasureComponent<Boolean>> ds_cond_2  
3013 then Dataset<T> ds_2}*  

```

3014 **else** Dataset<T> *ds_3*

3015 Constraints

- 3016 - *ds_cond_1*, *ds_1*, *ds_cond_2*, *ds_2* and *ds_3* must have the same IdentifierComponents, in
- 3017 name and type
- 3018 - *ds_cond_1* and *ds_cond_2* must have only one MeasureComponent<Boolean>
- 3019 - *ds_1*, *ds_2* and *ds_3* must have the same IdentifierComponents and
- 3020 MeasureComponents.

3021

3022 Semantics

3023 It takes in input a number of condition Datasets *ds_cond_1*, having exactly one
3024 MeasureComponent<Boolean> and, for each of them a Dataset *ds* to return in case of positive
3025 evaluation (**then**) of the condition. Besides it takes in input a default case (**else**) Dataset to be
3026 returned if all the previous conditions evaluate to false.

3027 All the involved Datasets must have the same IdentifierComponents, the returned Datasets
3028 must have the same MeasureComponents in name and type.

3029 Starting from *ds_cond_1*, for each row, if the MeasureComponent is *true*, it looks up in the
3030 corresponding Datasets (*ds_1*) all the rows for the corresponding values of the
3031 IdentifierComponents and returns them in *ds_4*. If the MeasureComponent is *false*, it looks up
3032 in the following *elseif* Dataset (*ds_cond_2*) for the corresponding values of the
3033 IdentifierComponents. If no row is found, the elaboration skips to the next row of *ds_cond_1*. If
3034 any rows are found and *ds_cond_2* is *true* for them, the corresponding **then** Dataset (*ds_2*) is
3035 returned; otherwise, the evaluation continues likewise, until the else part is reached (in case
3036 every previous conditional Datasets evaluate to *false*) and, if any matching rows are found in
3037 *ds_3* they are returned. Then the elaboration is repeated for all the rows in *ds_cond_1*.

3038 Parameters

3039 *ds_cond_1* – the condition Dataset to be evaluated. It drives the evaluation.

3040 *ds_1* - the Dataset containing the rows to be returned where *ds_cond_1* evaluates to *true* (the
3041 **then** part)

3042 *ds_cond_2* – the following condition Datasets to be evaluated where *ds_cond_1* evaluates to
3043 *false* (the **elseif** part).

3044 *ds_2* – the Dataset containing the rows to be returned where *ds_cond_2* evaluates to *true* (the
3045 **then** part of the **elseif**).

3046 *ds_3* – the Dataset containing the rows to be returned where all the previous condition
3047 Datasets evaluate to *false* (the **else** part).

3048 Returns

3049 *ds_4* – the Dataset to be returned, which is built according to the described semantics.

3050 Example

3051 Consider the following Datasets.

3052

population				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	M	5451780
2012	Belgium	Total	F	5643070
2012	Greece	Total	M	5449803
2012	Greece	Total	F	5673231
2012	Spain	Total	M	23099012
2012	Spain	Total	F	23719207
2012	France	Total	M	31616281
2012	France	Total	F	33671580
2012	Italy	Total	M	28726599
2012	Italy	Total	F	30667608
2012	Austria	Total	M	NULL
2012	Austria	Total	F	NULL

3053

unemp_rates_1				
TIME	GEO	AGE	SEX	RATE
2012	Spain	Total	F	25.8
2012	France	Total	F	NULL
2012	Italy	Total	F	20.9
2012	Austria	Total	M	6.3

3054

unemp_rates_2				
TIME	GEO	AGE	SEX	RATE
2012	Belgium	Total	M	0.12
2012	Greece	Total	M	22.5
2012	Spain	Total	M	23.7
2012	Austria	Total	F	NULL

3055

3056

The expression:

3057

```
ds_1 := if (population#SEX="F") #CONDITION
```

3058

```
then unemp_rates_1
```

3059 else unemp_rates_2

3060 will yield:

TIME	GEO	AGE	SEX	RATE
2012	Belgium	Total	M	0.12
2012	Greece	Total	M	22.5
2012	Spain	Total	M	23.7
2012	Spain	Total	F	25.8
2012	France	Total	F	NULL
2012	Italy	Total	F	20.9

3061

3062 population#SEX allows to consider SEX into the only *MeasureComponent*, which is
3063 compared with "F" by the operator "=". Correctly, it acts on the only
3064 *MeasureComponent* as POPULATION is temporarily not considered. The comparison
3065 returns CONDITION as the only *MeasureComponent*<Boolean>.

3066

3067 Thus, as the if operators requires a Dataset with a single
3068 *MeasureComponent*<Boolean>, # is applied again in order to isolate CONDITION_SEX as
3069 the only *MeasureComponent*<Boolean>.

3070

3071 **nvl**

3072 **Component version**

3073 Syntax

3074 Component<T> c :=

3075 **nvl**

3076 ({**component=**}Component<T> c_1, {**constant=**}Constant<T> const)

3077 Constraints

3078 None

3079 Semantics

3080 Returns c_1 if c_1 is not null, *const* otherwise.

3081 Examples

3082 Let us assume component C from a dataset.

3083 If C is null:

3084 A := nvl(C, 5) // A = 5

3085 If COMPX is not null and equal to 10:

```
3086     A := nvl(C, 5) // A = 10
```

3087 Dataset version

3088 Syntax

```
3089     Dataset<?+, MeasureComponent<T>+> ds_2 :=
3090     nvl
3091     ({dataset=}Dataset<?+, MeasureComponent<T>+> ds_1,
3092     {constant=} Constant<T> rep_value)
```

3093 Constraints

3094 All the MeasureComponents of *ds_1* must be of the same type (static).

3095 Semantics

3096 Takes in input a Dataset *ds_1* and a constant *rep_value* of the same type as the
3097 MeasureComponents of *ds_1*. It returns a second Dataset *ds_2* with the same
3098 IdentifierComponents of *ds_1*, the AttributeComponents (according to attribute propagation
3099 rule) and all the MeasureComponents. For each row of *ds_1* returns a row where the NULL
3100 values of the MeasureComponents are replaced with *rep_value*.

3101 Parameters

3102 *ds_1* – the Dataset to amend from NULL values

3103 *rep_value* – the value to be used to replace NULL values

3104 Returns

3105 *ds_2* a Dataset without NULL values for MeasureComponents

3106

3107 Example

3108

population				
TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	11094850
2012	Greece	Total	Total	11123034
2012	Spain	Total	Total	NULL
2012	Malta	Total	Total	417546
2012	Finland	Total	Total	5401267
2012	NULL	Total	Total	NULL

3109

3110

The expression

3111

```
ds_1 := nvl(population,0)
```

3112

applied against the previous Dataset, returns:

3113

TIME	GEO	AGE	SEX	POPULATION
2012	Belgium	Total	Total	11094850
2012	Greece	Total	Total	11123034
2012	Spain	Total	Total	0
2012	Malta	Total	Total	417546
2012	Finland	Total	Total	5401267
2012	NULL	Total	Total	0

3114

3115 Clauses (postfix operators)

3116 Clauses are operators adopting a postfix syntax, used to define expressions involving only
3117 Components. In particular:

3118 **rename** – allows to rename an existing Component

3119 **filter** – filters the rows in a Dataset according to a Component<Boolean> expression

3120 **keep** – chooses which Components to keep in or to drop from a Dataset

3121 **calc** – allows to calculate and add a new Component, alter the role of an existing one, calculate
3122 and modify the value of an existing one

3123 **attrcalc** – allows to define explicit calculation formulas for attributes

3124 **aggregate** – allows to perform data aggregations

3125 rename

3126 Syntax

```
3127 Dataset<?> ds_2 := Dataset<?> ds_1
3128 [rename Component<?> k as Constant<String> compName
3129                               {role=[MEASURE|IDENTIFIER|ATTRIBUTE]}
3130
3131   {, Component<?> k as Constant<String> compName
3132                               {role=[MEASURE|IDENTIFIER|ATTRIBUTE]}
3133   }*]
```

3134 Constraints

- 3135 - *k* is a Component expression that can have only Component literals of *ds_1* (static).
- 3136 - *role* can be one of: “MEASURE”, “IDENTIFIER”, “ATTRIBUTE” (static).

3137 Semantics

3138 It allows to change name and role of a Component of a Dataset.

3139 Takes in input a Dataset *ds_1* and returns a new Dataset *ds_2* with the same
3140 IdentifierComponents, MeasureComponents and Attributes according to attribute
3141 propagation rules.

3142 It renames each Component in *ds_1* that is mentioned in the operator with the new name
3143 given in *compName* and the role given in *role* variable. If *role* variable is not specified, the role
3144 is left unmodified. All the rows in *ds_1* are copied into *ds_2*.

3145

3146 Parameters

3147 *ds_2* – the input Dataset

3148 *k* – each Component to rename

3149 *compName* – the new name for each Component

3150 *role* – the new role for each Component

3151 Returns

3152 *ds_1* – the Dataset with renamed Components

3153 Example

3154 `ds_2 := ds_1[rename M1 as "I1" role IDENTIFIER]`

3155 The expression above renames MeasureComponent M1 into I1 and alters its role to
3156 IdentifierComponent.

3157

3158 **filter**

3159 Syntax

3160 `Dataset<?> ds_1 :=`
3161 `Dataset<?> ds_2`
3162 `[filter Component<Boolean> f]`

3163 Constraints

3164 - *f* is a Component expression over the Components of *ds_1* (static).

3165 Description

3166 Performs row-wise filtering. It takes in input a Dataset *ds_2* and returns a new one with the
3167 same IdentifierComponents, MeasureComponents and AttributeComponents (according to
3168 attribute propagation rules).

3169 The Component<Boolean> expression *f* is evaluated for each row of *ds_1*. A row is returned in
3170 the result, only if the corresponding evaluation of *f* yields *true*.

3171 Parameters

3172 - *ds_2* – the Dataset to filter
3173 - *f* – the Component expression defining the filter

3174

3175 Returns

3176 - *ds_1* – the filtered Dataset

3177

3178 Example

3179 In this example we perform filtering of a Dataset with population data and will retain only the
3180 rows about females.

3181

3182 `population1`

SEX	AGE	GEO	TIME	POPULATION
M	Y_LT15	BE	2013	970428
M	Y15-64	BE	2013	3678355
M	Y_GE65	BE	2013	838653
F	Y_LT15	BE	2013	927644
F	Y15-64	BE	2013	3625561
F	Y_GE65	BE	2013	1121001
M	Y_LT15	UK	2013	5757444
M	Y15-64	UK	2013	20748657
M	Y_GE65	UK	2013	4917238
F	Y_LT15	UK	2013	5488356
F	Y15-64	UK	2013	20915924
F	Y_GE65	UK	2013	6068452

3183

3184

3185

3186

The expression:

```
population1[filter SEX = "F"]
```

will yield the result:

SEX	AGE	GEO	TIME	POPULATION
F	Y_LT15	BE	2013	927644
F	Y15-64	BE	2013	3625561
F	Y_GE65	BE	2013	1121001
F	Y_LT15	UK	2013	5488356
F	Y15-64	UK	2013	20915924
F	Y_GE65	UK	2013	6068452

3187

3188 **keep**

3189 Syntax

3190

3191

3192

```
Dataset<?> ds_1 :=
```

```
Dataset<?> ds_2
```

```
[keep Component<Boolean> k {, Component<Boolean> k }*]
```

3193

Constraints

3194

3195

- f is a Component expression over the Components of *ds_1* containing only Component literals (i.e. names of the Components of *ds_1*) (static).

3196 Semantics

3197 Selects a subset of the IdentifierComponent and MeasureComponent, performing a kind of
3198 relational projection.

3199 It takes in input a Dataset *ds_2* and returns a new one with the IdentifierComponents and
3200 MeasureComponents selected in *k*.

3201 Special care must be paid in relation to how AttributeComponents are propagated. Keep
3202 clause overrides virality, so that an AttributeComponent will be present in the output if and
3203 only if it is present as a Component expression in keep. In practice, if an AttributeComponent
3204 is not mentioned in keep it will be dropped, otherwise it will be kept independently of its
3205 virality.

3206 If an AttributeComponent is kept in the result, its virality is not altered.

3207 Parameters

3208 *ds_2* – the input Dataset

3209 *k* – each Component to be kept in the output

3210 Returns

3211 *ds_1* – the output Dataset

3212 Example

3213 In this example we apply keep clause on the following Dataset.

3214 **population1**

SEX	AGE	GEO	TIME	POPULATION
M	Y_LT15	BE	2013	970428
M	Y15-64	BE	2013	3678355
M	Y_GE65	BE	2013	838653
F	Y_LT15	BE	2013	927644
F	Y15-64	BE	2013	3625561
F	Y_GE65	BE	2013	1121001
M	Y_LT15	UK	2013	5757444
M	Y15-64	UK	2013	20748657
M	Y_GE65	UK	2013	4917238
F	Y_LT15	UK	2013	5488356
F	Y15-64	UK	2013	20915924
F	Y_GE65	UK	2013	6068452

3215

3216 Thus, the expression:

3217 `ds_1 := population1[keep(SEX, GEO, POPULATION)]`

3218 will yield the result:

SEX	GEO	POPULATION
M	BE	970428
M	BE	3678355
M	BE	838653
F	BE	927644
F	BE	3625561
F	BE	1121001
M	UK	5757444
M	UK	20748657
M	UK	4917238
F	UK	5488356
F	UK	20915924
F	UK	6068452

3219

3220 **calc**

3221 Syntax

```
3222 Dataset<?> ds_2 :=
3223 Dataset<?> ds_1
3224 [calc Component<?> k as Constant<String> compName}
3225     {role [MEASURE | IDENTIFIER | ATTRIBUTE]
3226     {viral}}
3227
3228     {, Component<?> k as Constant<String> compName}
3229     {role [MEASURE | IDENTIFIER | ATTRIBUTE]
3230     {viral}}
3231     ]*]
```

3232 Constraints

- 3233 - *k* is a Component expression on *ds_1* components (static).
- 3234 - *role* can be one of : "MEASURE", "IDENTIFIER", "ATTRIBUTE".

3235 Semantics

3236 Takes in input a Dataset *ds_1* and returns a new Dataset *ds_2* with the same
3237 IdentifierComponents, MeasureComponents.

3238 It adds to *ds_2* a Component for each Component expression *k* specified in the clause,
3239 calculating it row-wise according to the Component expression.

3240 The added Component is named *compName* and is given a role (IdentifierComponent,
3241 MeasureComponent or AttributeComponent) according to *role* Constant. If the role is omitted,
3242 "MEASURE" is implied. If any *k* coincides with the name of an existing Component in *ds_1*
3243 (even with different type), the calculated one replaces the former, in name, value and type.

3244 Special care must be paid to the handling of AttributeComponents. If a Component expression
3245 has the same name as an existing AttributeComponent, the previous one is overridden,
3246 independently of its virality. In this sense, **calc** clause overrides virality. On the other hand, if
3247 no AttributeComponent expressions override an existing Component, it will be kept in the
3248 result, only if viral, with unaltered virality. In general, when an AttributeComponent is
3249 calculated, its virality can be set by the use of keyword **viral**. If it is omitted, the
3250 AttributeComponent is non viral by default. As a special case of this, a **calc** can be also used
3251 simply to alter the virality of an AttributeComponent.

3252 Parameters

3253 *ds_2* – the input Dataset

3254 *k* – each Component expression to be calculated

3255 *role* – each role to be assigned to calculated Components

3256 Returns

3257 *ds_1* – the output Dataset

3258 Example

3259 `ds_2 := ds_1[calc M1*M2/3 as "M4" role MEASURE]`

3260 The expression above calculates a MeasureComponent by combining the ones of the
3261 involved Datasets.

3262 `ds_2 := ds_1[calc M1-1 as "M1" role MEASURE, M1+M2 as M2, if M2>3`
3263 `then M2 else M3 as M3]`

3264 Like the preceding example, but with a conditional logic.

3265 `ds_2 := ds_1[calc A1 + A2 as "A3" role ATTRIBUTE viral]`

3266 The expression above calculates AttributeComponent A3 as a combination of A1 and
3267 A2.

3268

3269 attrcalc

3270 The default attribute propagation rule based on weights can be altered with user-defined
3271 formulas. It is possible to associate a calculation formula for a specific attribute in an
3272 expression with an *attrcalc* clause.

3273 Syntax

3274 `Dataset<?, AttributeComponent<?>> ds_2 :=`

3275 `Dataset<?> ds_1`

```

3276     [attrcalc AttributeComponent<?> k as Constant<String> compName
3277     {viral}
3278     {, AttributeComponent<?> k as Constant<String> compName {viral}}*
3279     ]

```

3280 Constraints

3281 - *k* is a Component expression on *ds_1* components or on components used to calculate
3282 *ds_1* properly qualified (static).

3283 Semantics

3284 Takes in input a Dataset *ds_1* (which, in general, can be a complex expression of type Dataset)
3285 and returns a new Dataset *ds_2* with the same IdentifierComponents, MeasureComponents.

3286 *ds_2* has an Attribute Component named *compName* for each AttributeComponent<?>
3287 expression *k* specified in the clause. *k* is a component expression, evaluated row-wise.

3288 Special care must be paid to the handling of AttributeComponents. If a Component expression
3289 has the same name as an existing AttributeComponent, the previous one is overridden,
3290 independently of its virality. In this sense, **attrcalc** clause overrides virality. On the other
3291 hand, if no AttributeComponent expressions override an existing Component, it will be kept in
3292 the result, only if viral, with unaltered virality. In general, when an AttributeComponent is
3293 calculated, its virality can be set by the use of keyword **viral**. If it is omitted, the
3294 AttributeComponent is non viral by default. As a special case of this, an **attrcalc** can be also
3295 used simply to alter the virality of an AttributeComponent.

3296 Parameters

3297 *ds_1* – the input Dataset

3298 *k* – each Component expression to be calculated

3299 *compName* – the name of the AttributeComponent to be calculated

3300 Returns

3301 *ds_1* – the output Dataset with the calculated attribute component

3302 Example 1

3303 *ds_2* := *ds_1*[attrcalc QUALITY+1 as QUALITY]

3304 Calculates *ds_2*, keeping the QUALITY Attribute Component in *ds_1*, but adding 1 to its value.

3305 *ds_1*

K1	K2	M1	QUALITY
1	A	1	1
2	B	3	2
3	C	5	3

3306 *ds_2*

K1	K2	M1	QUALITY
1	A	1	2
2	B	3	3
3	C	5	4

1	A	1	2
2	B	3	3
3	C	5	4

3307

3308 Example 2

3309 `ds_r := (ds_1 + ds_2)[attrcalc ds_1#QUALITY + ds_2#QUALITY as QUALITY]`

3310 The expression calculates `ds_r` as the sum of Datasets `ds_1` and `ds_2`. Besides, it calculates the
 3311 `QUALITY` attribute as the sum of the two.

3312

ds_1

K1	K2	M1	QUALITY
1	A	1	1
2	B	3	2
3	C	5	3

3313

ds_2

K1	K2	M1	QUALITY
1	A	6	2
2	B	7	3
3	C	8	4

3314

ds_r

K1	K2	M1	QUALITY
1	A	7	3
2	B	10	5
3	C	13	7

3315

3316 Example 3

3317 `ds_r := (ds_1 + ds_2)`

```
3318 [
3319   attrcalc if ds_1#QUALITY="A" and ds_2#QUALITY="B" then "C"
3320             elseif ds_1#QUALITY="K" and ds_2#QUALITY="K" then "M"
3321             else "Z" AS AGGREGATED_QUALITY
3322 ]
```

3323 The expression calculates the `AGGREGATED_QUALITY` attribute as a combination of `QUALITY`
 3324 Attribute Components of the operands according to a decision rule.

3325 In particular, if `ds_1` quality is "A" and `ds_2` quality is "B", then the `AGGREGATED_QUALITY` will
 3326 be "C". Else, if `ds_1` quality is "K" and `ds_2` quality is "K", then "M" is returned. Otherwise "Z" is
 3327 the `AGGREGATED_QUALITY` value.

3328

ds_1			
K1	K2	M1	QUALITY
1	A	1	A
2	B	3	B
3	C	5	K

3329

ds_2			
K1	K2	M1	QUALITY
1	A	6	B
2	B	7	A
3	C	8	K

3330

ds_r			
K1	K2	M1	QUALITY
1	A	7	C
2	B	10	Z
3	C	13	M

3331

3332 Example 4

3333 ds_r := (ds_1 + ds_2)[attrcalc ds_1#QUALITY_1 + ds_2#QUALITY_1 as
 3334 QUALITY_1, ds_2#QUALITY_2 as QUALITY_2]

3335 The expression sums two multi-measure Datasets and calculates two Attribute Components
 3336 with different formulas: QUALITY_1 is the sum of ds_1#QUALITY_1 and ds_2#QUALITY_1,
 3337 while QUALITY_2 is simply copied from ds_2.

3338

ds_1					
K1	K2	M1	M2	QUALITY_1	QUALITY_2
1	A	1	5	1	2
2	B	3	3	2	7
3	C	5	1	3	4

3339

ds_2					
K1	K2	M1	M2	QUALITY_1	QUALITY_2
1	A	6	1	2	1
2	B	7	1	3	3
3	C	8	1	4	1

3340

ds_r					
K1	K2	M1	M2	QUALITY_1	QUALITY_2
1	A	7	6	3	1
2	B	10	4	5	3
3	C	13	2	7	1

3341 aggregate

3342 Syntax

```

3343 Dataset<?> ds_2 :=
3344 Dataset ds_1<?+>, MeasureComponent<Numeric>>
3345 [aggregate [sum|avg|median|count|count_distinct|min|max] (
3346 {include NULLS} MeasureComponent<Numeric> aggrPart),
3347 { [sum|avg|median|count|count_distinct|min|max] (
3348 {include NULLS} MeasureComponent<Numeric> aggrPart ) }*]

```

3349 Constraints

- 3350 - *ds_1* must have only Numeric MeasureComponents (static).
- 3351 - If the aggregation clause is present, for each MeasureComponent, there must be an aggregation function (static).
- 3352 - *aggrPart* can contain only Component names (static).
- 3353 - # clause cannot be used (static).

3355 Description

3356 It takes in input a Dataset *ds_1* and applies an aggregation function to each
3357 MeasureComponent, grouping by the values of the IdentifierComponents. It returns a Dataset
3358 with the same IdentifierComponents, MeasureComponents and AttributeComponents
3359 (according to the attribute propagation rules).

3360 Many different aggregation functions are possible:

- 3361 - sum: algebraic sum of values (if the clause *include NULLS* is present and a NULL is
3362 present among the values, the aggregation evaluates to NULL. Otherwise, NULLs are
3363 considered 0 in the sum).

- 3364 - avg: average of the values (if the clause *include NULLS* is present and a NULL is present
- 3365 among the values, the aggregation evaluates to NULL. Otherwise, NULLs are not
- 3366 considered in the average).
- 3367 - median: calculates the median of the involved values (if the clause *include NULLS* is
- 3368 present and a NULL is present among the values, the aggregation evaluates to NULL.
- 3369 Otherwise, NULLs are not considered in the average).
- 3370 - count: counts the values (if the clause *include NULLS* is present, NULLS are considered
- 3371 in the count, otherwise they are excluded).
- 3372 - count_distinct: counts the distinct values, excluding the duplicates (if the clause *include*
- 3373 *NULLS* is present, NULLS are considered in the count, otherwise they are excluded.
- 3374 Notice that if NULLS are considered, each NULL contributes to the count).
- 3375 - min: calculates the minimum value (if the clause *include NULLS* is present, NULL is
- 3376 returned only if it is the only value to aggregate).
- 3377 - max: calculates the maximum value (if the clause *include NULLS* is present, NULL is
- 3378 returned only if it is the only value to aggregate).

3379 Parameters

3380 *ds_1* – the Dataset to aggregate

3381 *aggrPart* – the MeasureComponents to aggregate on

3382 Returns

3383 *ds_2* – the aggregated Dataset

3384 Example 1

3385 Consider the following Dataset:

3386

3387 **Population 1**

SEX	AGE	GEO	TIME	POPULATION
M	Y_LT15	BE	2013	90
M	Y15-64	BE	2013	100
M	Y_GE65	BE	2013	150
F	Y_LT15	BE	2013	130
F	Y15-64	BE	2013	120
F	Y_GE65	BE	2013	110
M	Y_LT15	UK	2013	70
M	Y15-64	UK	2013	80
M	Y_GE65	UK	2013	120
F	Y_LT15	UK	2013	40
F	Y15-64	UK	2013	400
F	Y_GE65	UK	2013	500

3388

3389

The expression

3390

```
ds_1 := population1[keep[SEX,GEO,TIME,POPULATION][aggregate
```

3391

```
sum(POPULATION) ]
```

3392

computes the total population by country, sex and reference year, i.e. we aggregate

3393

over age classes. It yields the following result:

3394

the result would then be:

3395

SEX	GEO	TIME	POPULATION
M	BE	2013	340
F	BE	2013	360
M	UK	2013	270
F	UK	2013	940

3396

3397 Example 2

3398

In this example we calculate the average in Dataset datasetA, with microdata about the

3399

monthly income of individuals.

3400

datasetA

PERS_ID	YEAR	GEO	SEX	INCOME
DE001	2011	DE	M	2340
DE002	2011	DE	M	1850
DE003	2011	DE	M	2170
DE004	2011	DE	M	0
DE005	2011	DE	F	2200
DE006	2011	DE	F	1850
DE007	2011	DE	F	1840
DE008	2011	DE	F	1680
FR001	2011	FR	M	1150
FR002	2011	FR	M	3450
FR003	2011	FR	M	2360
FR004	2011	FR	M	1830
FR005	2011	FR	F	1940
FR006	2011	FR	F	1150
FR007	2011	FR	F	NULL

3401 ds_1 := datasetA[keep(GEO,SEX,INCOME)][aggregate avg(INCOME)]

3402

ds_1		
GEO	SEX	INCOME
DE	M	1590.0
DE	F	1892.5
FR	M	2197.5
FR	F	1545.0

3403

3404 It is easy to see in this example that rows with NULL values are ignored in the calculation of
3405 the mean for consistency with other operators.

3406 Example 3

3407 In this example we count the distinct entries of the DatasetB by YEAR, GEO, SEX.

3408

DatasetB

PERS_ID	YEAR	GEO	SEX	OCCUPATION ³	INCOME
DE001	2011	DE	M	1	2340
DE002	2011	DE	M	4	1850
DE003	2011	DE	M	2	2170
DE004	2011	DE	M	5	0
DE005	2011	DE	F	NULL	2200
DE006	2011	DE	F	8	1851
DE007	2011	DE	F	5	1840
DE008	2011	DE	F	2	1680
FR001	2011	FR	M	5	1153
FR002	2011	FR	M	1	3450
FR003	2011	FR	M	1	2360
FR004	2011	FR	M	4	1830
FR005	2011	NULL	F	8	1940
FR006	2011	FR	F	5	1150
FR007	2011	FR	F	7	NULL
FR008	2011	NULL	F	NULL	NULL

³ The occupation column is coded according to ISCO-08 one digit.

3409

The expression

3410

DatasetB [keep (YEAR, GEO, SEX, OCCUPATION)] [aggregate count
3411 OCCUPATION]

3412

would yield the result:

YEAR	GEO	SEX	COUNT
2011	DE	M	4
2011	DE	F	4
2011	FR	M	4
2011	NULL	F	2
2011	FR	F	2

3413

3414

Example 4

3415

In this example we count the distinct entries of the DatasetB by YEAR and GEO.

3416

DatasetB

PERS_ID	YEAR	GEO	SEX	OCCUPATION ⁴	INCOME
DE001	2011	DE	M	1	2340
DE002	2011	DE	M	4	1850
DE003	2011	DE	M	2	2170
DE004	2011	DE	M	5	0
DE006	2011	DE	F	8	1851
DE007	2011	DE	F	5	1840
DE008	2011	DE	F	2	1680
FR001	2011	FR	M	5	1153
FR002	2011	FR	M	1	3450
FR003	2011	FR	M	1	2360
FR004	2011	FR	M	4	1830
FR006	2011	FR	F	5	1150
FR007	2011	FR	F	7	NULL

3417

3418

The expression:

3419

ds_1 := DatasetB [keep (YEAR, GEO, OCCUPATION)] [aggregate
3420 count (OCCUPATION)]

⁴ The occupation column is coded according to ISCO-08 one digit.

3421 would yield the result:

YEAR	GEO	OCCUPATION
2011	DE	5
2011	FR	4

3422 Note that despite there are 8 rows for DE only 5 are counted in the result because
3423 there are five distinct values and the NULL is ignored in the result.

3424 Example 5

3425 Let us start again from DatasetA in example 2.

3426 The expression:

3427 `ds_1 := datasetA[keep(GEO,SEX,INCOME)][aggregate min(INCOME)]`

3428 will yield the result:

3429

ds_1		
GEO	SEX	INCOME
DE	M	0.0
DE	F	1680.0
FR	M	1150.0
FR	F	1150.0

3430

3431 Equally, the expression:

3432 `ds_1 := datasetA[keep(GEO,SEX,INCOME)][aggregate max(INCOME)]`

3433 will yield the result:

ds_1		
GEO	SEX	INCOME
DE	M	2340.0
DE	F	2200.0
FR	M	3450.0
FR	F	1940.0

3434

3435

3436 Example 6

3437 Consider the Dataset population1.

3438

population1

SEX	AGE	GEO	TIME	POPULATION
M	Y_LT15	BE	2013	100
M	Y15-64	BE	2013	200
M	Y_GE65	BE	2013	150
F	Y_LT15	BE	2013	320
F	Y15-64	BE	2013	NULL
F	Y_GE65	BE	2013	340
M	Y_LT15	UK	2013	400
M	Y15-64	UK	2013	110
M	Y_GE65	UK	2013	130
F	Y_LT15	UK	2013	320
F	Y15-64	UK	2013	1000
F	Y_GE65	UK	2013	500

3439

3440

The expression

3441

```
ds_1 := population1[keep[SEX,GEO,TIME,POPULATION][aggregate
```

3442

```
sum(include NULLS POPULATION)]
```

3443

computes the total population by country, sex and reference year. The “include NULLS” clause specifies that NULLS must be considered in the aggregation. Therefore for the female, Belgian aggregate, POPULATION is NULL.

3444

3445

3446

ds_1			
SEX	GEO	TIME	POPULATION
M	BE	2013	450
F	BE	2013	NULL
M	UK	2013	640
F	UK	2013	1820

3447

3449 Labour Force Survey and VTL

3450 Introduction

3451 In this section, the Validation and Transformation Language is used to describe some
3452 operations needed in a Labour Force Questionnaire to validate micro-data.

3453 In the remainder of the section, the micro-data Dataset is named **DS** while the Components in
3454 the Dataset are named as the name of the question in the questionnaire and referred to with
3455 the *DS#DS_FIELD* notation.

3456 For example in the question below, which belongs to Dataset DS:

3457 **SG18B.** In what year did you come to live in Italy the first time?

3458 the name of the question is SG18B, hence it corresponds to Component SG18B in Dataset DS1
3459 and, as usual, can be referred to with DS#SG18B.

3460 There are three additional fields that belong to Dataset DS, although they are not directly part
3461 of the questionnaire: **REG** and **MUN**, which are the region and the municipality where the
3462 respondent lives respectively, and **SURVEY_YEAR**, which is the year when the survey takes
3463 place. These fields are part of the Dataset but are not part of the questionnaire, in the sense
3464 that their values are not provided by the respondent, as they are implicitly defined by the
3465 place and the time of the survey.

3466 The remainder of this section is organized as follows:

3467 In the first part, *incompatibility rules*, some of the rules used to validate the micro-data
3468 Dataset resulting from the survey, are defined. In these cases, operators are applied to one or
3469 more Components of the same Dataset; this means that there are no comparisons with other
3470 Datasets. In order to facilitate the interpretation of the rules, the questions of the
3471 questionnaire, used for the definition of the rules, are also reported.

3472 Every rule includes:

- 3473 • **Operators** - the VTL operators involved in the rule
- 3474 • **Questions** - the questions of the questionnaire from which the rule is derived
- 3475 • **Textual rule**- a textual description of rule

- **VTL rule**- the definition of the rule using the VTL language. In order to make rules more clear, they have been divided into sub-conditions. The resulting Dataset, deriving from the combination of such condition is always named **DS_R** in the example. Operators are reported in bold.

The second part, **Domains of variables**, shows some VTL rules to check variable domains. For the sake of clarity, also in this case the questions have been reported.

The third part, **Aggregation**, contains some examples of possible aggregations of micro-data to obtain a simple Dataset of macro-data (i.e. employed by sex). The examples are based on two aggregation rules used to define employed and unemployed people on the basis of micro-data Dataset.

Compatibility rules

- 1) **Operators**: check, not, LESS THAN, not EQUAL, and

Questions:

SG18B. In what year did you come to live in Italy the first time?

- *Year* |_|_|_|_|
- *Don't know* |0|9|9|7|

SG18F. Since when have you been living in Italy without leaving the Country for one year or more?

- *Year* |_|_|_|_|
- *Don't know* |0|9|9|7|

Textual rule: The year declared in SG18F cannot be less than of the year in which he came to Italy for the first time

The rule above results from the combination of two conditions below:

SG18B_SG18F_NotNull :

the answer to SG18F question is different from "Don't know".

➤ **SG18F_NotNull** := ds#SG18F <> 997

SG18F_LT_SG18B (SG18F is less than SG18B):

The year declared in SG18F is less than the year declared on SG18B

➤ **SG18F_LT_SG18B**:= ds[rename SG18F as
"CONDITION"]#CONDITION < ds[rename SG18B as
"CONDITION"]#CONDITION

VTL rule:

3504
3505 **ds_r:=check (not (SG18B_SG18F_NotNull and SG18F_LT_SG18B) ,**
3506 **imbalance (DS#SG18B-DS#SG18F) ,**
3507 **erlevel (8) ,**
3508 **errorcode (2001) ,**
3509 **all)**

3510 where error code 2001 corresponds to 'Compatibility error for questions SG18B ,
3511 SG18F'

3512 2) **Operators:** check, not, LESS THAN, not EQUAL, SUBTRACTION, ISNULL

Questions:

3514 **SG18B.** In what year did you come to live in Italy the first time?

- 3515 • Year |_|_|_|_|
3516 • Don't know |0|9|9|7|

3517 **SG18F.** Since when have you been living in Italy without leaving the Country for one year or
3518 more?

- 3519 • Year |_|_|_|_|
3520 • Don't know |0|9|9|7|

3521 **SG20.** Would you like to tell me how old you are?

- 3522 • Age |_|_|_|

3523
3524 **Textual rule:** Age cannot be less than the years of residence in Italy

3525
3526 The rule above results from the combination of three conditions below:

SG18B_SG18F_NotNull:

The answer to the SG18B and SG18F questions is different from "Don't know".

```
➤ SG18B_SG18F_NotNull := ds#SG18F <> 997 and ds#SG18B <> 997
```

SG20_NotNull:

The answer to the SG20 questions is different from null.

```
➤ SG20_NotNull := not isnull(ds#SG20)
```

SG20_LT_RSD (SG20 Less Than Residence):

The year answered to the question SG20 is less than the residence year, obtained as difference between the year of the survey less the year of coming in Italy.

```
➤ SG20_LT_RSD := ds[rename SG20 as "CONDITION"]#CONDITION <
  ds[rename SURVEY_YEAR as "CONDITION"]#CONDITION -
  ds[rename SG18F as "CONDITION"]#CONDITION
```

3528

VTL rule:

3529

```
ds_r:=check(not(SG18B_SG18F_NotNull and SG20_NotNull and
```

3530

```
SG20_LT_RSD),
```

3531

```
imbalance(ds#SG20-( ds#SURVEY_YEAR - DS#SG18F )),
```

3532

```
erlevel(8),
```

3533

```
errorcode(2002),
```

3534

```
all)
```

3535

3536

3) **Operators:** if...then...else, check, not, LESS THAN, not EQUAL, or,and, ADDITION,

3537

SUBTRACTION,IN

3538

3539

SG23_SG19_LT_Y14 (SG20-SG19 Less than 14 years):

The difference between the date of the marriage and the date of birth is less than 14

```
➤ SG23_SG19_LT_Y14:= ds[rename SG23 as  
"CONDITION"]#CONDITION - ds[rename SG19_YEAR5as  
"CONDITION"]#CONDITION) < 14
```

AGE_LT_Y14 :

The difference between age at the moment of the interview and the age of marriage is less than 14

```
➤ AGE_LT_Y14:= ds[rename SG23 as "CONDITION"]#CONDITION -  
ds[rename SURVEY_YEAR as "CONDITION"]#CONDITION) +  
ds[rename SG21 as "CONDITION"]#CONDITION < 14
```

VTL rule:

```
ds_r:=check(not((SG19_NotNull and SG23_NotNull and  
SG23_SG19_LT_Y14))  
or  
(not( SG19_Null and SG23_NotNull and not  
AGE_LT_Y14)),  
Imbalance(if (ds#SG19_YEAR=997) then (ds#SG21)  
else (ds#SG19_YEAR),  
erlevel(8),  
errorcode(2003),  
all)
```

4) Operators: check, not IN, EQUAL

⁵ In this case the answer to the question SG19 is stored in three different fields: SG19_YEAR for the year, SG19_MONTH for the month, SG19_DAY for the day.

3577

Questions:

3578

SG24. Which is your highest school qualification?

3579

• *No qualification* 1

3580

• *Primary school* 2

3581

• *Diploma of lower secondary education* 3

3582

• *2-3-year upper secondary school professional qualification not allowing admission to university* 4

3583

• *Diploma of upper secondary education (4-5-years) allowing admission to University Music conservatory certificate/National dance academy certificate* 5

3584

• *Academy of Fine Arts Licence, Higher Institute for Applied Arts Diploma, National Academy of Drama Diploma, National Dance Academy (advanced course) Diploma, Music Conservatory (advanced course) Diploma, Institute for Interpreters and Translators qualification, Archivist, Diplomatic and Palaeography School qualification*

3585

• *Art and Music degree* 6

3586

• *University education undergraduate diploma* 7

3587

• *University first degree (3years programmes)* 8

3588

• *University second degree (2years programmes)* 9

3589

• *University degree (4years and more old and new programmes)* 10

3590

3591

3592

3593

3594

3595

3596

SG24B. Do you have a postgraduate degree or doctorate?

3597

• *Master (university or Art and Music) first level* 1

3598

• *Master (university or Art and Music) second level* 2

3599

• *Post graduate degree* 3

3600

• *Doctorate (PhD)* 4

3601

• *No postgraduate degree* 5

3602

3603

3604

Textual rule:The educational qualifications (SG24) equal to 7 or 8 does not allow to access to a post graduate degree.

3605

3606

The rule above results from the combination of two conditions below:

SG24_IN_7_8 (SG24 take values 7 or 8):

The highest school qualification is 7 or 8

➤ **SG24_IN_7_8** := ds#SG24 **in** (7, 8)

SG24B_EQ_3 (SG24B is equal to 3):

The interviewer has a post graduate degree

➤ **SG24B_EQ_3** := ds#SG24B = 3

```

3607 VTL rule:
3608 ds_r:=check(SG24_IN__7_8 and SG24B_EQ_3,
3609             erlevel(8) ,
3610             errorcode(2004) ,
3611             all)

```

5) **Operators:** substr, and, equals, IN, check

3614 **Questions :**

C1. Do you work:

- **Read all the answering items**
- *as an employee* 1

with an:

- *employer-coordinated freelance work contract (on specific project or not)* 2
- *occasional work contract* 3

as self-employed:

- *running a business* 4
- *as a professional* 5
- *as an own-account worker* 6
- *as a family worker* 7
- *as a member of producers' co-operative* 8

C12. Occupation code

[ISCO]

|_|_|_|_|_|

3632 **Textual rule:** a self-employed (as running a business, as a professional or as an own-
3633 account worker) usually is not involved in an occupation group 4⁶ (professions
3634 executive office).

3636 The rule above results from the combination of two conditions below:

⁶ The occupation group is calculated as the first character of the ISCO08 code

C1_IN_4_6 (C1 take values from 4 to 6)

The type of work is 4 or 5 or 6

➤ **C1_IN_4_6** := ds#C1 **in** (4, 5, 6)

C12_EQ_4 (The first character of C12 is equal to 4)

The code of the profession has length 4 digit

➤ **C12_LEN_NEQ_4**:= substr(DS#C12,1)=4

3637

VTL rule:

3638

ds_r:= **check** (**not**(C1_IN_4_6 **and** C12_LEN_NEQ_4) ,

3639

imbalance (C12) ,

3640

errorcode (2005) ,

3641

all)

3642

3643

6) **Operators:** check, and, not EQUAL, ADDITION, SUBTRACTION, MULTIPLICATION

3644

Questions :

3645

3646

SG19. Would you like to tell me your date of birth?

3647

• *Day* *|||* • *Don't know* *|9|9|7|*

3648

• *Month* *|||* • *Don't know* *|9|9|7|*

3649

• *Year* *||||* • *Don't know* *|0|9|9|7|*

3650

3651

3652

3653
 3654
 3655
 3656
 3657
 3658
 3659
 3660
 3661
 3662
 3663
 3664
 3665
 3666
 3667
 3668
 3669
 3670
 3671
 3672
 3673
 3674
 3675
 3676

E2 When was the last time you worked? Please indicate the conclusion year	
<i>If s/he did not work in the reference week (B2=1) or s/he is an occasional worker or a seasonal work (B3=994,995)</i>	
• Year	_ _ _ _
• Don't know	997_
E4. In which month? (month in which the responder stopped work)	
• Month	_ _
• Don't know	997_
E14. What is the main reason why you stopped working?	
• Retirement (contributory, non-contributory pension)	1_
• Dismissed or made redundant (also due to bankruptcy or closing down of the company)	2_
• Job of limited duration (including occasional and seasonal work)	3_
• Illness, personal health problems	4_
• Maternity, birth of a child	5_
• Looking after children and/or other incapacitated persons	6_
• Education or professional training	7_
• Compulsory military or community service	8_
• Other family reasons (except for maternity, care of children or of other people)	9_
• Other reasons (specify)_____	996_

3677 **Textual rule:** The respondent cannot have started the military service before the age of
 3678 18.

3679

3680 The rule above results from the combination of five conditions below:

E4_E2_NotNull

The answers E2 and E4 are different from "Don't know"

➤ **E4_E2_NotNull** := ds#E4<>997 and ds#E2<>997

E14_EQ_8

The answers E14 is equal to 8

➤ **E14_EQ_8** := ds#E14=8

SG19_Y_M_NotNull

The answers SG19_YEAR and SG19_MONTH are different from "Don't know"

➤ **SG19_Y_M_NotNull**:=ds#SG19_YEAR<>997 and ds#SG19_MONTH<>997

YEARS_MS (Years military service)

The years of the interviewer before he started the military service

➤ **YEARS_MS** := ds[rename E2 as "CONDITION"]#CONDITION -
ds[rename SG19_YEAR as "CONDITION"] #CONDITION)

MONTHS_MS (Months military service)

The difference of months between the birth month and the month when the military service started

➤ **MONTHS_MS**:= ds[rename SG19_MONTH as "CONDITION"]#CONDITION
- ds[rename E4 as "CONDITION"] #CONDITION)

VTL rule:

```
3681 ds_r:=check      (not((E4_E2_NotNull      and      E14_EQ_8      and  
3682      SG19_Y_M_NotNull) and (YEARS_MS *12 + MONTHS_MS)<216) ,  
3683      errorCode(2006) ,  
3684      all)
```

3681
3682
3683
3684
3685
3686
3687
3688
3689

3690 7) **Operators:** check, ISNULL, EQUAL, and

3691 **Questions**

3692
3693 **I5.** 12 months ago (on REFMONTH of Year before REFYEAR), were you:

- 3694
- 3695 • *Employed* 1
 - 3696 • *Unemployed looking for new employment* 2
 - 3697 • *Looking for first employment* 3
 - 3698 • *Fulfilling domestic tasks* 4
 - 3699 • *Student* 5
 - 3700 • *Retired* 6
 - 3701 • *Disabled for work* 7
 - 3702 • *In other condition* 9
- 3703

3704 **C1.** Do you work:

- 3705 • *as an employee* 1
- 3706 with an:
- 3707 • *employer-coordinated freelance work contract (on specific project or not)* 2
 - 3708 • *occasional work contract* 3
- 3709 as self-employed:
- 3710 • *running a business* 4
 - 3711 • *as a professional* 5
 - 3712 • *as an own-account worker* 6
 - 3713 • *as a family worker* 7
 - 3714 • *as a member of producers' co-operative* 8
- 3715

3716 **Textual rule:** the respondent has declared to be unable to work then he cannot answer
3717 question C1.

3718 The rule above results from the combination of two conditions below:

I5_EQ_7

The interviewer is "Disabled for work"

➤ **I5_EQ_7** := ds#I5=7

C1_IsNotNull

Question C1 is not null

➤ **C1_IsNotNull :=** (DS#C1 not ISNULL)

3719

3720

VTL rule:

3721

ds_r := check(not (I5_EQ_7 and C1_IsNotNull), imbalance(ds#C1), errorcode(2007),

3722

all)

3723

Domains of the variables

3724

1) **Operators:** check, RANGE

3725

SG4. At first I am going to ask you some information about your household. How many people permanently live in this household, including yourself (except for possible domestic servants or tenants)?

3726

3727

3728

FOR THE INTERVIEWER: Please enter the number of people living in this household on Sunday

3729

• *Number of household members*

|||

3730

Textual rule: the number of components cannot be more than 15

3731

VTL rule:

3732

ds_r := check(ds#SG4 between 0 and 15, all)

3733

3734

2) **Operators:** check, RANGE

3735

SG20. Would you like to tell me how old you are?

3736

[YEARBIR]

3737

• *Age* ||||

3738

Textual rule: the age cannot be more than 120 years

3739

VTL rule:

3740

ds_r := check(ds#SG20 between 0 and 120, all)

3741

3742

3) **Operators:** and, or, equal to, greater than, less than

3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768

C31A. How many hours did you work on average in a week? Please consider the last 4 week excluding meal breaks and travel time between home and the place of work.

- *Number of hours* |_|_|_|
- *Don't know* 997 |_|_|_|

Textual rule: the usual working hours of the last 4 weeks for the main activity cannot be more than 150 or they can assume 997

VTL rule:

```
ds_r := check((ds#C31A>0 and ds#C31A<120) or ds#C31A=997, all)
```

4) **Operators:** EXISTS_IN

Codelist **ISCO08** represents ISCO 2008 classification.

C12. Occupation code

Coding based on the annex "Classification of Occupations"

- _____ |_|_|_|_|

Textual rule: the value of the occupation code must be equal to the ISCO 2008 code

VTL rule:

```
ds_r := check_codelist(ds#C12 , ISCO08#CODE, all)
```

3769 **Aggregation**

3770 In the Labour Force Survey there are some rules needed to define if the interviewed is
 3771 occupied or not. These rules refer to a subset of questions reported below:

3772

3773

3774

3775

B1. In the week from Monday to Sunday, did you work for at least one hour? Please consider paid work or the work you expect to be paid for or unpaid work only if in some relative's business

3776

• Yes 1

3777

3778

• No 2

3779

3780

• Permanently disabled for work 3

3781

3782

B3. What is the main reason why you did not work in that week?

3784

• Lay-off (ordinary or extraordinary) 1

3785

• Slack work for economical and/or technical reasons (except for lay-off) 2

3786

• Labour dispute 4

3787

• Bad weather 5

3788

• Illness, personal health problems, accident 6

3789

• Annual holidays 7

3790

• Bank holidays during the week 8

3791

• Variable working hours or flexitime (e.g. compensation leave) 9

3792

• Vertical part-time 10

3793

• Education or training not organized by the employer 11

3794

• Compulsory maternity leave 12

3795

• Optional maternity/paternity leave until the child's eighth birthday (parental leave) 13

3796

• Family reasons (no including compulsory maternity/paternity leave and parental leave) 14

3797

• No chances for more work 15

3798

• Occasional worker 994

3799

• Seasonal worker as employee (e.g. lifeguard, fruit gatherer, waiter in mountain during the winter, etc.) 995

3800

• Other reasons (specify) 996

3801

3802

B4. Do you work:

3805

as an employee 1

3806

with an:

3807

• employer-coordinated freelance work contract (on specific project or not) 2

3808

• occasional work contract 3

3809

as a self-employed:

3810

• running a business 4

3811

• as a professional 5

3812

• as an own-account worker 6

3813

• as a family worker 7

3814

• as a member of producers' co-operative 8

3815

3816

3817

3818

3819

3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871

B4A.α. Do you have an employment contract?

B4A.β. Do you have an employment contract or an employer-coordinated freelance work contract?

- Yes, *an employment contract* 1
- Yes, *an employer-coordinated freelance work contract* 2
- No 3

B4bis.α. The employer-coordinated freelance work contract (on a specific project or not) provides the obligation for the employer to pay for the insurance contribution to the INPS? Does your employer pay for this?

B4bis.β. The occasional work contract provides the payment of a withholding tax. Does your employer pay for the withholding tax on your contract?

- Yes 1
- No 2

B6. Will this period of absence last less or more than three months from start to finish?

If s/he is in parental leave (B3=13) consider only the period of parental leave excluding the period of compulsory maternity/paternity leave (if the parental leave comes straight after the period of compulsory absence)

- *Till three months* 1
- *Three months or more* 2

B7. Is this period of absence paid at least partially?

- Yes, *50% or more* 1
- Yes, *less than 50%* 2
- No, *it is not paid* 3

B8. Is it a leave period?

- Yes 1
- No 2
- *Don't know* 997
- *Don't answer* 998

B9. Is your job paid as family worker?

- Yes 1
- No 2

B10. In the week from Monday to Sunday, was your work contract still in force, i.e. its expiry date was after Monday ...?

- Yes 1
- No 2

B11. Is your working activity interrupted at the moment (e.g. for professional updating, for restructuring of premises, seasonal closing) or is it definitively finished?

- *Interrupted at the moment* 1
- *Finished* 2

3872

3873

3874

VTL rules to define if the interviewed is employed if one of the following conditions is respected:

B1_B3

The answer to the question B1 is “yes” and the answer to B3 is from 5 to 9 or equal to 12

➤ **B1_B3** := (DS#B1=1) or (DS#B3 IN (3,5,6,7,8,9,12))

B4_B4A_B6_B7

The answer to B4 is 1 or the answer to B4A is 1 and the answer to B6 is 1 or to B7 is 1

➤ **B4_B4A_B6_B7** := ((ds#B4=1 or ds#B4A=1) and (ds#B6=1 or ds#B7=1))

B4bis_B10

The answer to the B4bis question is 1 and the answer to the question B10 is 1

➤ **B4bis_B10** := (ds#B4bis=1 and ds#B10=1)

B4_B4A_B11

The answer to the B4 is from 4 to 6 or the answer to B4 is 8 and B4A is 3 and B11 is equal to 11

➤ **B4bis_B10** := ((ds#B4 in (4,5,6) or (ds#B4=8 and ds#B4A=3)) and ds#B11=1)

B4_B4A_B6_B9

The answer to the B4 is 7 and the answer to B4A is 3 and B6 is 1 or B9 is equal to 1

➤ **B4bis_B10** := (ds#B4=7 and ds#B4A=3 and (ds#B6=1 or ds#B9=1))

3875 **ds_r:= (B1_B3) or (B4_B4A_B6_B7) or (B4bis_B10) or (B4_B4A_B11) or**
3876 **(B4_B4A_B6_B9)**

3877 **Aggregation:** Number of occupied by region

3878 **ds_r:= get(DS, Keep(REG),**

3879 **filter (Result), count_distinct)**

3880

3881 1) **Aggregation:** Number of occupied by region and municipality

3882 **ds_r:= get(DS, Keep(REG,MUN),**

3883 **filter (Result),**

3884 **count_distinct)**

3885 2) **Aggregation:** Number of occupied by region and sex

SG11. Sex of the member	
• <i>Male</i>	1 _
• <i>Female</i>	2 _

3886 **ds_r:= get(DS, Keep(REG, SG11),**

3887 **filter (Result), count_distinct)**