

WG3:W22-008

ISO/IEC JTC 1/SC 32

Date: 2022-07-07

IWD 39075:202y(E)

ISO/IEC JTC 1/SC 32/WG 3

The United States of America (ANSI)

Information technology — Database languages — GQL

Technologies de l'information — Langages de base de données — GQL

Document type: International Standard
Document subtype: Informal Working Draft (IWD)
Document stage: IWD-20
Document language: English
Document name: 39075_1IWD25-GQL_2022-07

Edited by: Stefan Plantikow (Ed.) and Stephen Cannan (Assoc. Ed.)

PDF rendering performed by XEP, courtesy of RenderX, Inc.



Copyright notice

This ISO document is a working draft or a committee draft and is copyright-protected by ISO. While the reproduction of working drafts or committee drafts in any form for use by participants in the ISO standards development process is permitted without prior permission from ISO, neither this document nor any extract from it may be reproduced, stored or transmitted in any form for any other purpose without prior written permission from ISO.

Requests for permission to reproduce for the purpose of selling it should be addressed as shown below or to ISO's member body in the country of the requester.

*ANSI Customer Service Department
25 West 43rd Street, 4th Floor
New York, NY 10036
Tele: 1-212-642-4980
Fax: 1-212-302-1286
Email: storemanager@ansi.org
Web: www.ansi.org*

Reproduction for sales purposes may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

Contents	Page
Foreword.....	xiv
Introduction.....	xv
1 Scope.....	1
2 Normative references.....	2
3 Terms and definitions.....	4
3.1 Introduction to terms and definitions.....	4
3.2 General terms and definitions.....	4
3.3 Graph terms and definitions.....	6
3.4 GQL-environment terms and definitions.....	8
3.5 GQL-catalog terms and definitions.....	11
3.6 Procedure terms and definitions.....	12
3.7 Procedure syntax terms and definitions.....	16
3.8 Value terms and definitions.....	19
3.9 Type terms and definitions.....	21
3.10 Temporal terms and definitions.....	23
3.11 Definitions taken from ISO/IEC 14651:2019.....	24
4 Concepts.....	25
4.1 Use of terms.....	25
4.2 GQL-environments and their components.....	25
4.2.1 General description of GQL-environments.....	25
4.2.2 GQL-agents.....	26
4.2.3 GQL-implementations.....	26
4.2.3.1 Introduction to GQL-implementations.....	26
4.2.3.2 GQL-clients.....	27
4.2.3.3 GQL-servers.....	27
4.2.4 Basic Security Model.....	28
4.2.4.1 Principals.....	28
4.2.4.2 Authorization identifiers.....	28
4.2.5 GQL-catalog.....	28
4.2.5.1 General description of the GQL-catalog.....	28
4.2.5.2 GQL-directories.....	29
4.2.5.3 GQL-schemas.....	30
4.2.6 GQL-data.....	31
4.3 GQL-objects.....	31
4.3.1 General introduction to GQL-objects.....	31
4.3.2 GQL-objects held by static sites.....	32
4.3.3 GQL-objects held by dynamic sites.....	32
4.3.4 Graphs.....	33

4.3.4.1	Introduction to graphs.	33
4.3.4.2	Graph descriptors.	34
4.3.5	Binding tables.	35
4.3.6	Libraries.	36
4.3.7	Type objects.	37
4.3.8	Procedure objects.	37
4.4	Values.	37
4.4.1	General information about values.	37
4.4.2	Property values and supported property value types.	37
4.4.3	Reference values.	38
4.4.4	Values classified according to their use of references.	38
4.5	GQL-sessions.	38
4.5.1	General description of GQL-sessions.	38
4.5.2	Session contexts.	39
4.5.2.1	Introduction to session contexts.	39
4.5.2.2	Session context creation.	40
4.5.2.3	Session context modification.	40
4.6	GQL-transactions.	40
4.6.1	General description of GQL-transactions.	40
4.6.2	Transaction demarcation.	41
4.6.3	Transaction isolation.	41
4.6.4	Encompassing transaction belonging to an external agent.	42
4.7	GQL-requests.	42
4.7.1	General description of GQL-requests.	42
4.7.2	GQL-request contexts.	42
4.7.2.1	Introduction to GQL-request contexts.	42
4.7.2.2	GQL-request context creation.	43
4.7.2.3	GQL-request context modification.	43
4.7.3	Working objects.	43
4.7.4	Execution stack.	44
4.8	Execution contexts.	44
4.8.1	General description of execution contexts.	44
4.8.2	Execution context creation.	45
4.8.3	Execution context modification.	46
4.8.4	Execution outcomes.	46
4.9	Diagnostics.	47
4.9.1	Introduction to diagnostics.	47
4.9.2	Conditions.	47
4.9.3	GQL-status object.	49
4.10	Procedures and commands.	49
4.10.1	General description of procedures and commands.	49
4.10.2	Procedures.	50
4.10.2.1	General description of procedures.	50
4.10.2.2	Procedure descriptors.	50
4.10.2.3	Procedure signatures.	50
4.10.2.4	Procedure signature descriptor.	51
4.10.2.5	Type signature descriptor.	51

4.10.2.6	Procedure execution.....	51
4.10.2.7	Procedures classified by type of computation.....	52
4.10.2.8	Procedures classified by type of provisioning.....	52
4.10.3	Commands.....	52
4.10.3.1	General description of commands.....	52
4.10.3.2	Command execution.....	53
4.10.4	GQL-procedures.....	53
4.10.4.1	Introduction to GQL-procedures.....	53
4.10.4.2	Variables and parameters.....	53
4.10.4.3	Statements.....	54
4.10.4.4	Statements classified by use of the current working graph.....	54
4.10.4.5	Statements classified by function.....	54
4.10.4.6	Scope of names.....	54
4.10.5	Operations.....	56
4.10.5.1	Introduction to operations.....	56
4.10.5.2	Operations classified by execution outcome.....	56
4.10.5.3	Operations classified by type of side effects.....	57
4.11	Graph pattern matching.....	57
4.11.1	Summary.....	57
4.11.2	Paths.....	58
4.11.3	Path patterns.....	58
4.11.4	Graph pattern variables.....	60
4.11.5	References to graph pattern variables.....	61
4.11.6	Path pattern matching.....	63
4.11.7	<path mode>.....	64
4.11.8	Selective <path search prefix>.....	64
4.11.9	<match mode>.....	65
4.12	Data types.....	65
4.12.1	General introduction to data types.....	65
4.12.2	Naming of predefined value types and associated base types.....	66
4.12.3	Data type descriptors.....	67
4.12.4	Data type terminology.....	67
4.12.5	Properties of distinct.....	68
4.13	Type hierarchy outline.....	68
4.13.1	Introduction to type hierarchy outline.....	68
4.13.2	The any type.....	68
4.13.3	The any object type.....	68
4.13.4	The any value type.....	68
4.13.5	Union types.....	69
4.13.6	The any property value type.....	69
4.13.7	The nothing type.....	69
4.14	Graph types.....	69
4.14.1	Introduction to graph types.....	69
4.14.2	Graph type descriptors.....	71
4.14.3	Graph element types.....	71
4.14.3.1	Graph element base type.....	72
4.14.3.2	Node base type.....	72

4.14.3.3	Node types.....	72
4.14.3.4	Node type descriptor.....	73
4.14.3.5	Edge base type.....	73
4.14.3.6	Edge types.....	73
4.14.3.7	Edge type descriptor.....	74
4.14.3.8	Property types.....	75
4.14.3.8.1	Introduction to property types.....	75
4.14.3.8.2	Property type descriptor.....	75
4.15	Binding table types.....	75
4.16	Value types.....	76
4.16.1	Constructed types.....	76
4.16.1.1	Introduction to constructed types.....	76
4.16.1.2	List types.....	76
4.16.1.2.1	List types.....	76
4.16.1.2.2	Regular lists.....	77
4.16.1.2.3	Distinct lists.....	77
4.16.1.3	Path type.....	77
4.16.1.4	Map type.....	78
4.16.1.5	Record type.....	78
4.16.2	Reference value types.....	79
4.16.3	Predefined value types.....	80
4.16.3.1	Boolean types.....	80
4.16.3.2	Character string types.....	81
4.16.3.2.1	Introduction to character strings.....	81
4.16.3.2.2	Collations.....	81
4.16.3.3	Byte string types.....	82
4.16.3.4	Numeric types.....	82
4.16.3.4.1	Introduction to numbers.....	82
4.16.3.4.2	Characteristics of numbers.....	83
4.16.3.4.3	Binary exact numeric types.....	85
4.16.3.4.4	Decimal exact numeric types.....	86
4.16.3.4.5	Approximate numeric types.....	86
4.16.3.5	Temporal types.....	87
4.17	Sites.....	88
4.17.1	General description of sites.....	88
4.17.2	Assignment.....	88
4.17.3	Nullability.....	88
5	Notation and conventions.....	89
5.1	Notation taken from The Unicode® Standard	89
5.2	Notation.....	89
5.3	Conventions.....	90
5.3.1	Specification of syntactic elements.....	90
5.3.2	Specification of the Information Graph Schema.....	91
5.3.3	Use of terms.....	92
5.3.3.1	Syntactic containment.....	92
5.3.3.2	Terms denoting rule requirements.....	92
5.3.3.3	Rule evaluation order.....	92

5.3.3.4	Conditional rules.....	93
5.3.3.5	Syntactic substitution.....	94
5.3.4	Descriptors.....	94
5.3.5	Subclauses used as subroutines.....	95
5.3.6	Index typography.....	95
5.3.7	Feature ID and Feature Name.....	96
6	GQL-requests.....	97
6.1	<GQL-request>.....	97
6.2	<GQL-program>.....	100
7	Session management.....	101
7.1	<session set command>.....	101
7.2	<session remove command>.....	103
7.3	<session clear command>.....	104
7.4	<session close command>.....	105
8	Transaction management.....	106
8.1	<start transaction command>.....	106
8.2	<end transaction command>.....	107
8.3	<transaction characteristics>.....	108
8.4	<rollback command>.....	109
8.5	<commit command>.....	110
9	Procedures.....	111
9.1	<procedure specification>.....	111
9.2	<query specification>.....	114
9.3	<procedure body>.....	115
10	Variable and parameter declaration and definition.....	118
10.1	Binding variable and parameter declaration and definition.....	118
10.2	Graph variable and parameter declaration and definition.....	120
10.3	Binding table variable and parameter declaration and definition.....	123
10.4	Value variable and parameter declaration and definition.....	126
11	Object and object type expressions.....	129
11.1	Introduction to object and object type expressions.....	129
11.2	<graph expression>.....	130
11.3	<graph type expression>.....	131
11.4	<binding table type>.....	133
12	Statements.....	135
12.1	<statement>.....	135
12.2	<call procedure statement>.....	136
12.3	Statement classes.....	138
13	Catalog-modifying statements.....	140
13.1	<linear catalog-modifying statement>.....	140
13.2	<create schema statement>.....	142
13.3	<drop schema statement>.....	143
13.4	<create graph statement>.....	144
13.5	<graph specification>.....	147
13.6	<drop graph statement>.....	149

13.7	<create graph type statement>.....	151
13.8	<graph type specification>.....	153
13.9	<node type definition>.....	155
13.10	<edge type definition>.....	157
13.11	<label set definition>.....	163
13.12	<property type set definition>.....	164
13.13	<property type definition>.....	164
13.14	<drop graph type statement>.....	166
13.15	<call catalog-modifying procedure statement>.....	168
14	Data-modifying statements.....	169
14.1	<linear data-modifying statement>.....	169
14.2	<do statement>.....	171
14.3	<insert statement>.....	172
14.4	<set statement>.....	175
14.5	<remove statement>.....	179
14.6	<delete statement>.....	181
14.7	<call data-modifying procedure statement>.....	183
15	Query statements.....	184
15.1	<composite query statement>.....	184
15.2	<composite query expression>.....	185
15.3	<linear query expression>.....	188
15.4	<linear query statement>.....	189
15.5	Data-reading statements.....	191
15.5.1	<match statement>.....	191
15.5.2	<call query statement>.....	193
15.6	Data-transforming statements.....	194
15.6.1	<mandatory statement>.....	194
15.6.2	<optional statement>.....	196
15.6.3	<filter statement>.....	198
15.6.4	<let statement>.....	199
15.6.5	<aggregate statement>.....	201
15.6.6	<for statement>.....	203
15.6.7	<order by and page statement>.....	206
15.7	Result projection statements.....	208
15.7.1	<primitive result statement>.....	208
15.7.2	<return statement>.....	210
15.7.3	<select statement>.....	215
15.7.4	<project statement>.....	217
16	Common elements.....	218
16.1	<use graph clause>.....	218
16.2	<at schema clause>.....	220
16.3	Named elements.....	221
16.4	<type signature>.....	223
16.5	<graph pattern>.....	225
16.6	<path pattern prefix>.....	234
16.7	<path pattern expression>.....	239

16.8	<simple graph pattern>.....	250
16.9	<label expression>.....	252
16.10	<graph pattern quantifier>.....	254
16.11	<simplified path pattern expression>.....	256
16.12	<where clause>.....	262
16.13	<procedure call>.....	263
16.14	<inline procedure call>.....	264
16.15	<named procedure call>.....	266
16.16	<yield clause>.....	269
16.17	<group by clause>.....	271
16.18	<order by clause>.....	273
16.19	<aggregate function>.....	274
16.20	<sort specification list>.....	277
16.21	<limit clause>.....	280
16.22	<offset clause>.....	281
17	Object references.....	282
17.1	Schema references.....	282
17.2	Graph references.....	284
17.3	Graph type references.....	287
17.4	Binding table references.....	290
17.5	Procedure references.....	293
17.6	Function references.....	296
17.7	<catalog object reference>.....	299
17.8	<qualified object name>.....	302
17.9	<url path parameter>.....	304
17.10	<external object reference>.....	305
17.11	<element reference>.....	306
18	Predicates.....	308
18.1	<search condition>.....	308
18.2	<predicate>.....	309
18.3	<comparison predicate>.....	311
18.4	<exists predicate>.....	314
18.5	<null predicate>.....	316
18.6	<normalized predicate>.....	317
18.7	<directed predicate>.....	318
18.8	<labeled predicate>.....	319
18.9	<source/destination predicate>.....	320
18.10	<all_different predicate>.....	322
18.11	<same predicate>.....	323
19	Value expressions.....	324
19.1	<value specification>.....	324
19.2	<value expression>.....	327
19.3	<boolean value expression>.....	329
19.4	<numeric value expression>.....	332
19.5	<value expression primary>.....	334
19.6	<numeric value function>.....	335

19.7	<string value expression>.....	342
19.8	<string value function>.....	345
19.9	<datetime value expression>.....	352
19.10	<datetime value function>.....	353
19.11	<duration value expression>.....	357
19.12	<duration value function>.....	359
19.13	<list value expression>.....	361
19.14	<list value function>.....	363
19.15	<list value constructor>.....	365
19.16	<map value constructor>.....	366
19.17	<record value constructor>.....	368
19.18	<field>.....	370
19.19	<property reference>.....	371
19.20	<value query expression>.....	372
19.21	<case expression>.....	373
19.22	<cast specification>.....	376
19.23	<element_id function>.....	386
20	Lexical elements.....	387
20.1	<literal>.....	387
20.2	<value type>.....	397
20.3	<property value type>.....	410
20.4	<field type>.....	411
20.5	Names and identifiers.....	412
20.6	<token> and <separator>.....	415
20.7	<GQL terminal character>.....	424
21	Additional common rules.....	428
21.1	Satisfaction of a <label expression> by a label set.....	428
21.2	Machinery for graph pattern matching.....	430
21.3	Evaluation of a <path pattern expression>.....	436
21.4	Evaluation of a selective <path pattern>.....	441
21.5	Applying bindings to evaluate an expression.....	444
21.6	Determination of identical values.....	447
21.7	Determination of distinct values.....	448
21.8	Equality operations.....	449
21.9	Ordering operations.....	450
21.10	Collation determination.....	451
21.11	Result of value type combinations.....	452
22	Diagnostics.....	455
23	Status codes.....	457
23.1	GQLSTATUS.....	457
24	Conformance.....	460
24.1	Introduction to conformance.....	460
24.2	Minimum conformance.....	460
24.3	Conformance to features.....	461
24.4	Requirements for GQL-programs.....	461
24.4.1	Introduction to requirements for GQL-programs.....	461

24.4.2	Claims of conformance for GQL-programs.....	461
24.5	Requirements for GQL-implementations.....	462
24.5.1	Introduction to requirements for GQL-implementations.....	462
24.5.2	Claims of conformance for GQL-implementations.....	462
24.5.3	Extensions and options.....	462
24.6	Requirements for graphs.....	462
24.6.1	Introduction to requirements for graphs.....	462
24.6.2	Claims of conformance for graphs.....	462
24.7	GQL Flagger.....	463
24.8	Implied feature relationships.....	463
24.9	Syntactic transformations applied before Conformance Rules in SQL/PGQ.....	465
Annex A (informative) GQL Conformance Summary	466	
Annex B (informative) Implementation-defined elements	476	
Annex C (informative) Implementation-dependent elements	491	
Annex D (informative) GQL feature taxonomy	494	
Annex E (informative) Maintenance and interpretation of GQL	498	
Annex F (informative) Differences with SQL	499	
Annex G (informative) Graph serialization format	504	
Bibliography	505	
Index	506	

Tables

Table		Page
1	Symbols used in BNF.....	89
2	Conversion of simplified syntax delimiters to default edge delimiters.....	258
3	Truth table for the AND Boolean operator.....	330
4	Truth table for the OR Boolean operator.....	330
5	Truth table for the IS Boolean operator.....	331
6	Command codes.....	455
7	GQLSTATUS class and subclass codes.....	457
8	Implied feature relationships.....	463
9	Syntactic transformations applied before Conformance Rules in SQL/PGQ.....	465
10	Feature taxonomy for optional features.....	494

Figures

Figure	Page
1 Components of a GQL-environment	25
2 Components of a GQL-catalog	29

Foreword

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

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

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

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

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

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

This is the first edition of ISO/IEC 39075.

Any feedback or questions on this document should be directed to the user's national standards body. www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

This document defines the data structures and basic operations on the GQL-catalog and GQL-data for the GQL language.

The GQL language is a composable declarative database language for working with property graphs that can be used both as an independent language as well as in conjunction with SQL or other languages.

As a declarative language, GQL is intended to allow multiple, different implementation strategies (e.g., by pattern matching using relational algebra, linear algebra, or automata theory) using different storage representations (e.g., index-free adjacency storage, classic tabular storage techniques, matrix representations) and targeting various classes of workloads (e.g., OLTP, OLAP).

Information technology — Database languages — GQL

1 Scope

** Editor's Note (number 1) **

WG3:W05-020 suggests that this clause should be reviewed for adherence to the NWIP scope, adherence to current ISO directives, completeness, and readability. See [Possible Problem GQL-135](#).

The GQL language provides functional capabilities for:

- Querying, modifying, and projecting property graphs.
- Querying, modifying, and projecting property graph views.
- Transforming binding tables.
- Working with primitive data values.
- Working with nested collections and maps of data values.
- Composing requests from procedures and commands and composing optionally parameterized procedures from nested procedures, statements, patterns, expressions, and similar constructs.
- Managing named GQL-directories, GQL-schemas and catalog objects in the GQL-catalog.
- Working with catalog objects such as property graphs, property graph types (comprising graph element and constraint definitions), user-defined and built-in procedures, queries.
- Session management.
- Transaction demarcation.
- Interacting with other languages and systems.
- Handling errors.
- Reporting diagnostic information.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 8601-1:2019, *Date and time — Representations for information interchange — Part 1: Basic rules*

ISO 8601-2:2019, *Date and time — Representations for information interchange — Part 2: Extensions*

ISO/IEC 9075-2:202x, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*

ISO/IEC/IEEE 9945, *Information technology — Portable Operating System Interface (POSIX®) Base Specifications*

ISO/IEC 10646, *Information technology — Universal Multi-Octet Coded Character Set (UCS)*

ISO/IEC 14651:2019, *Information technology — International string ordering and comparison — Method for comparing character strings and description of the common template tailorable ordering*

IEEE Std 754:2019, *IEEE Standard for Floating-Point Arithmetic*

Berners-Lee, T., Fielding, R., Masinter, L.. *Uniform Resource Identifier (URI): Generic Syntax* [online]. Wilmington, Delaware, USA: Network Working Group, January 2005 . Available at <http://www.ietf.org/rfc/rfc3986.txt>

Duerst, M., Suignard, M.. *Internationalized Resource Identifiers (IRI)* [online]. Wilmington, Delaware, USA: Network Working Group, January 2005 . Available at <http://www.ietf.org/rfc/rfc3987.txt>

IANA. *Time Zone Database* [online]. Los Angeles, California, USA: Internet Assigned Numbers Authority, The Time Zone Database (often called tz or zoneinfo) contains code and data that represent the history of local time for many representative locations around the globe. It is updated periodically to reflect changes made by political bodies to time zone boundaries, UTC offsets, and daylight-saving rules. Available at <https://www.iana.org/time-zones>

Kuhn, Markus. *Coordinated Universal Time with Smoothed Leap Seconds (UTC-SLS)* [online]. University of Cambridge: IETF, January 2006 . Available at <https://tools.ietf.org/html/draft-kuhn-leapsecond-00>

The Unicode Consortium. *The Unicode Standard (Information about the latest version of the Unicode standard can be found by using the "Latest Version" link on the "Enumerated Versions of The Unicode Standard" page.)* [online]. Mountain View, California, USA: The Unicode Consortium, Available at <http://www.unicode.org/versions/enumeratedversions.html>

The Unicode Consortium. *Unicode Collation Algorithm* [online]. Mountain View, California, USA: The Unicode Consortium, Available at <http://www.unicode.org/reports/tr10/>

The Unicode Consortium. *Unicode Normalization Forms* [online]. Mountain View, California, USA: The Unicode Consortium, Available at <http://www.unicode.org/reports/tr15/>

The Unicode Consortium. *Unicode Identifier and Pattern Syntax* [online]. Mountain View, California, USA: The Unicode Consortium, Available at <http://www.unicode.org/reports/tr31/>

van Kesteren, A.. *URL Living Standard* [online]. [Place of publication unknown]: WHATWG, Available at <https://url.spec.whatwg.org>

3 Terms and definitions

** Editor's Note (number 2) **

WG3:W05-020 suggests that, although much of this clause taken from SQL/PGQ and SQL Foundation, and is not expected to be controversial, it should nevertheless be reviewed. See Possible Problem [GQL-136].

3.1 Introduction to terms and definitions

For the purposes of this document, the terms and definitions given in ISO 8601-1:2019, ISO 8601-2:2019, ISO/IEC 14651:2019 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- IEC Electropedia: available at <http://www.electropedia.org/>
- ISO Online browsing platform: available at <http://www.iso.org/obp>

In this document, the definition of a verb defines every voice, mood, and tense of that verb.

** Editor's Note (number 3) **

The following definitions also have been slightly modified from SQL/PGQ to reflect

- consistent use of “x pattern” and of “x pattern variable”
- preferred use of “element pattern variable” instead of “primary graph pattern variable”
- use of refined “x pattern” definitions

These changes need to be taken into account when adopting content from SQL/PGQ. They should ultimately flow back into SQL/PGQ.

3.2 General terms and definitions

3.2.1

atomic, adj.

incapable of being subdivided

Note 1 to entry: The antonym is *composite* (3.2.2).

3.2.2

composite, adj.

comprising distinguishable elements

Note 1 to entry: The antonym is *atomic* (3.2.1).

« WG3:BER-010 P00-USA-303 »

3.2.3

identify

reference without ambiguity

3.2.4

implementation-defined, adj.

possibly differing between GQL-implementations, but specified by the implementor for each particular GQL-implementation

3.2.5

implementation-dependent, adj.

possibly differing between GQL-implementations, but not required to be specified by the implementor for any particular GQL-implementation

3.2.6

identifier

value (3.8.1) by which something is identified

3.2.7

object

<“X” object> *thing* of some type “X” that is separately identifiable, has a definition, and possibly, content

Note 1 to entry: The content of an object is mutable unless explicitly defined otherwise.

Note 2 to entry: Objects are not *values* (3.8.1).

3.2.8

descriptor

<“X” descriptor> coded description of a *GQL-object* (3.4.17) that defines the *metadata* of a *GQL-object* (3.4.17) of a specified type

Note 1 to entry: A descriptor includes all information about its GQL-object that is required by a conforming GQL-implementation.

Note 2 to entry: See Subclause 5.3.4, “Descriptors”.

3.2.9

persistent, adj.

continuing to exist indefinitely, until destroyed deliberately

3.2.10

dictionary

<“X” dictionary> mapping defined by a collection of identifier-value pairs

Note 1 to entry: Different pairs never have the same identifier.

« WG3:BER-010 P00-USA-303 »

3.2.11

hold

<“X” in a dictionary> contain one or more key-value pairs whose value is “X”

« WG3:BER-094R1 deleted one definition »

3.2.12

set

unordered collection of distinguishable elements

3.2.13

sequence

ordered collection of elements that are not necessarily distinguishable

« WG3:BER-038R1 deleted one definition »

3.2.14

cardinality

<of a collection> number of elements in that collection

Note 1 to entry: Those elements need not necessarily have distinguishable values. The objects to which this concept applies include graphs, binding tables and the values of constructed types.

3.2.15**temporary**, adj.

lasting for only a limited period of time

3.2.16**variable**<“X” variable> *identifier* (3.2.6) assigned to a *site* (3.7.10)

Note 1 to entry: A variable represents a site.

3.3 Graph terms and definitions

3.3.1**graph**

property graph

data *object* (3.2.7) comprising zero or more *labels* (3.3.19), zero or more *properties* (3.3.20), zero or more *nodes* (3.3.11) and zero or more *edges* (3.3.12)

Note 1 to entry: In the context of working with property graphs, the term graph is commonly used in the real world as a shorthand for property graph. Without judgment or prejudice, this document prefers the term graph. Additionally, the term property graph is only used to establish the right context in introductory text.

3.3.2**multigraph***graph* (3.3.1) that allows more than one *edge* (3.3.12) connecting two *nodes* (3.3.11)**3.3.3****directed graph***graph* (3.3.1) in which every *edge* (3.3.12) is directed**3.3.4****undirected graph***graph* (3.3.1) in which every *edge* (3.3.12) is an *undirected edge* (3.3.15)**3.3.5****mixed graph***graph* (3.3.1) that allows both *directed edges* (3.3.14) and *undirected edges* (3.3.15)**3.3.6****empty graph***graph* (3.3.1) with zero *nodes* (3.3.11) and zero *edges* (3.3.12)**3.3.7****path***sequence* (3.2.13) of zero or an odd number of *graph elements* (3.3.10)Note 1 to entry: A path with one or more graph elements always starts and ends with a *node* (3.3.11) and alternates between nodes and *edges* (3.3.12) such that each edge resides in the path between its *endpoints* (3.3.16). See Subclause 4.11.2, “Paths”, for a fuller discussion of paths.

Note 2 to entry: A path may comprise a single node.

Note 3 to entry: A node or an edge may be contained multiple times in a path, including via self-loops.

3.3.8

subpath

path (3.3.7) fully contained in another path

Note 1 to entry: A subpath may be identical to its containing path.

Note 2 to entry: A subpath cannot be longer than its containing path.

Note 3 to entry: A subpath may be the zero-node path.

« WG3:RKE-031 »

3.3.9

zero-node path

empty sequence of *graph elements* (3.3.10)

Note 1 to entry: The term “empty path” is avoided because that is commonly used for a sequence of length 1 having one *node* (3.3.11) node and no *edges* (3.3.12).

3.3.10

graph element

node (3.3.11) or *edge* (3.3.12)

3.3.11

node

vertex

fundamental unit of which a *graph* (3.3.1) is formed

Note 1 to entry: Plural: nodes or vertices.

Note 2 to entry: Both terms, node and vertex, are used in the real world to mean the same. Without judgment or prejudice, this document uses only the term node. In BNF productions, wherever the keyword NODE is allowed, the keyword VERTEX can be used instead.

3.3.12

edge

relationship

connection between two *nodes* (3.3.11)

Note 1 to entry: Both terms, edge and relationship, are used in the real world to denote the same concept. Without judgment or prejudice, this document uses only the term edge. In BNF productions, wherever the keyword EDGE is allowed, the keyword RELATIONSHIP can be used instead.

3.3.13

directionality

<edge> information regarding whether an *edge* (3.3.12) is a *directed edge* (3.3.14) or an *undirected edge* (3.3.15)

3.3.14

directed edge

edge (3.3.12) that distinguishes one of its *endpoints* (3.3.16) as its *source node* (3.3.17) and one of its endpoints as its *destination node* (3.3.18)

Note 1 to entry: A directed edge expresses a relationship that is asymmetric.

Note 2 to entry: The antonym is *undirected edge* (3.3.15).

3.3.15**undirected edge**

edge (3.3.12) that does not distinguish between its *endpoints* (3.3.16)

Note 1 to entry: An undirected edge expresses a relationship that is necessarily symmetric.

Note 2 to entry: The antonym is *directed edge* (3.3.14).

3.3.16**endpoint**

incident node

<edge> one of the two *nodes* (3.3.11) connected by an *edge* (3.3.12)

Note 1 to entry: Both endpoints of an edge may be the same node.

« WG3:BER-081 deleted one definition »

3.3.17**source node**

start node

node (3.3.11) that is distinguished as the source of a *directed edge* (3.3.14)

3.3.18**destination node**

end node

node (3.3.11) that is distinguished as the destination of a *directed edge* (3.3.14)

3.3.19**label**

identifier (3.2.6) associated with a *graph* (3.3.1), a *node* (3.3.11), or an *edge* (3.3.12)

« WG3:BER-040R3 »

3.3.20**property**

<GQL-object> pair comprising a name and a *value* (3.8.1)

Note 1 to entry: The value of a property is a value of its *property type* (3.8.8).

« WG3:BER-073 deleted one Editor's Note. »

3.4 GQL-environment terms and definitions

3.4.1**GQL-environment**

milieu in which the *GQL-catalog* (3.5.1) and *GQL-data* (3.5.4) exists and *GQL-requests* (3.4.11) are executed

Note 1 to entry: See Subclause 4.2, “GQL-environments and their components”.

3.4.2**GQL-server**

processor capable of executing a *GQL-request* (3.4.11) that was submitted by a *GQL-client* (3.4.4) and delivering the outcome of that *execution* (3.6.7) back to the GQL-client in accordance with the rules and definitions of the GQL language

Note 1 to entry: See Subclause 4.2.3.3, “GQL-servers”.

3.4.3**principal**

object (3.2.7) that represents a user within a GQL-implementation

Note 1 to entry: See Subclause 4.2.4.1, “Principals”.

3.4.4**GQL-client**

processor capable of establishing a connection to a *GQL-server* (3.4.2) on behalf of a *GQL-agent* (3.4.5) that is authenticated to represent a *principal* (3.4.3)

Note 1 to entry: See Subclause 4.2.3.2, "GQL-clients".

3.4.5**GQL-agent**

independent process that causes the *execution* (3.6.7) of *procedures* (3.6.24) and *commands* (3.6.36)

Note 1 to entry: See Subclause 4.2.2, "GQL-agents".

« WG3:BER-040R3 »

3.4.6**GQL-session**

time period in which consecutive *GQL-requests* (3.4.11) are executed by a *GQL-client* (3.4.4) on behalf of a *GQL-agent* (3.4.5) with the same *GQL-environment* (3.4.1)

Note 1 to entry: See Subclause 4.5, "GQL-sessions".

3.4.7**session context**

context associated with a *GQL-session* (3.4.6) in which multiple *GQL-requests* (3.4.11) are executed consecutively

3.4.8**current session context**

session context (3.4.7) associated with the *GQL-session* (3.4.6) of the currently executing *GQL-request* (3.4.11)

« WG3:BER-040R3 »

3.4.9**session parameter**

pair comprising a name and a value that is defined in a *session context* (3.4.7)

3.4.10**GQL-transaction**

logical unit of independent *atomic* (3.2.1) *execution* (3.6.7)

Note 1 to entry: See Subclause 4.6, "GQL-transactions".

3.4.11**GQL-request**

request source (3.4.12) and *request parameters* (3.4.13)

Note 1 to entry: See Subclause 4.7, "GQL-requests".

3.4.12**request source**

character string (3.8.19) containing the source text of a GQL-program

« WG3:BER-040R3 »

3.4.13**request parameter**

pair comprising a name and a value that is defined in a *GQL-request context* (3.4.14)

3.4.14

GQL-request context

context that augments a *session context* (3.4.7) in which an individual *GQL-request* (3.4.11) is executed

Note 1 to entry: See Subclause 4.7.2, "GQL-request contexts".

3.4.15

current request context

GQL-request context (3.4.14) associated with the currently executing *GQL-request* (3.4.11)

3.4.16

execution stack

push-down stack of *execution contexts* (3.6.9) associated with a *GQL-request* (3.4.11)

3.4.17

GQL-object

object (3.2.7) capable of being manipulated directly by the *execution* (3.6.7) of a *GQL-request* (3.4.11)

Note 1 to entry: See Subclause 4.3, "GQL-objects".

3.4.18

data object

GQL-object (3.4.17) comprising data

3.4.19

primary object

independently definable *GQL-object* (3.4.17)

Note 1 to entry: A primary object may be defined separately from, but may also be contained as a component of another object.

« WG3:BER-040R3 »

3.4.20

secondary object

GQL-object (3.4.17) necessarily defined as a component of other *GQL-objects* (3.4.17)

« WG3:BER-040R3 deleted two definitions »

3.4.21

static object type

object type (3.9.14) comprising *static objects* (3.4.22)

3.4.22

static object

GQL-object (3.4.17) assignable to a *static site* (3.4.23)

« WG3:BER-040R3 »

3.4.23

static site

site (3.7.10) holding *static* (3.4.24) instances

3.4.24

static, adj.

always determinable at request submission time

Note 1 to entry: The antonym is *dynamic* (3.4.26).

« WG3:BER-040R3 deleted two definitions »

« WG3:BER-040R3 »

3.4.25**dynamic site**

site (3.7.10) holding *dynamic* (3.4.26) instances

3.4.26**dynamic**, adj.

possibly determinable at *execution* (3.6.7) time only

Note 1 to entry: The antonym is *static* (3.4.24).

« WG3:BER-040R3 deleted one definition »

3.4.27**original**, adj.

<GQL-object> defined independently

Note 1 to entry: The antonym is *derived* (3.4.28).

3.4.28**derived**, adj.

<GQL-object> defined by a computation from other *objects* (3.2.7)

Note 1 to entry: The antonym is *original* (3.4.27).

3.5 GQL-catalog terms and definitions

3.5.1**GQL-catalog**

persistent (3.2.9), hierarchically-organized collection of *GQL-directories* (3.5.3) and *GQL-schemas* (3.5.5)

Note 1 to entry: See Subclause 4.2.5, “GQL-catalog”.

« WG3:BER-061 »

3.5.2**GQL-catalog root**

GQL-directory (3.5.3) or a *GQL-schema* (3.5.5) that is the root of the *GQL-catalog* (3.5.1)

3.5.3**GQL-directory**

persistent (3.2.9) *dictionary* (3.2.10) of *GQL-directories* (3.5.3) and *GQL-schemas* (3.5.5)

Note 1 to entry: See Subclause 4.2.5.2, “GQL-directories”.

3.5.4**GQL-data**

data described by *GQL-schemas* (3.5.5) in the *GQL-catalog* (3.5.1) — data that is under the control of a GQL-implementation in a *GQL-environment* (3.4.1)

Note 1 to entry: See Subclause 4.2.6, “GQL-data”.

3.5.5**GQL-schema**

persistent (3.2.9) *dictionary* (3.2.10) of *primary catalog objects* (3.5.9)

Note 1 to entry: See Subclause 4.2.5.3, “GQL-schemas”.

« WG3:BER-040R3 »

3.5.6**library**

primary catalog object (3.5.9) comprising a *dictionary* (3.2.10) of catalog objects

« WG3:BER-082R1 deleted one definition »

3.5.7**type object***catalog object* (3.5.8) reifying the existence of a *data type* (3.9.2)

« WG3:BER-040R3 deleted one definition »

3.5.8**catalog object***GQL-object* (3.4.17) directly or indirectly defined in the *GQL-catalog* (3.5.1)**3.5.9****primary catalog object***GQL-object* (3.4.17) that is both a *primary object* (3.4.19) and a *catalog object* (3.5.8)**3.5.10****visible**

<GQL-object> capable of being referenced according to effective access control rules

3.5.11**home schema**default *GQL-schema* (3.5.5) associated with a *principal* (3.4.3)**3.5.12****home graph**default *graph* (3.3.1) associated with a *principal* (3.4.3)

3.6 Procedure terms and definitions

3.6.1**side effect**change caused during the *execution* (3.6.7) of a *GQL-request* (3.4.11) that is detectable by the execution of another *operation* (3.6.8) as part of the execution of the same or another GQL-request**3.6.2****catalog-modifying**, adj.<side effect> modifying the *GQL-catalog* (3.5.1)**3.6.3****session-modifying**, adj.<side effect> modifying the *GQL-session* (3.4.6) and its context**3.6.4****transaction-modifying**, adj.<side effect> manipulating *GQL-transactions* (3.4.10)**3.6.5****data-populating**, adj.<side effect> initially populating the content of newly created *data objects* (3.4.18) using *data-modifying* (3.6.6) *operations* (3.6.8)**3.6.6****data-modifying**, adj.<side effect> modifying the content of *data objects* (3.4.18)**3.6.7****execution**computation of a result that may cause *side effects* (3.6.1)

3.6.8

operation

identifiable action carried out by an *execution* (3.6.7)

3.6.9

execution context

context comprising a *dictionary* (3.2.10) of objects that is associated with and manipulated by the *execution* (3.6.7) of *operations* (3.6.8)

Note 1 to entry: See Subclause 4.8, "Execution contexts".

3.6.10

current execution context

execution context (3.6.9) of the currently executing *operation* (3.6.8) of the currently executing *procedure* (3.6.24) or *command* (3.6.36)

3.6.11

execution outcome

component of an *execution context* (3.6.9) representing the outcome of an *execution* (3.6.7)

Note 1 to entry: See Subclause 4.8.4, "Execution outcomes".

3.6.12

current execution outcome

execution outcome (3.6.11) of the *current execution context* (3.6.10)

« WG3:BER-060 »

3.6.13

successful outcome

<execution context> *execution outcome* (3.6.11) representing the successful and complete *execution* (3.6.7) of the last *operation* (3.6.8) executed in the *current execution context* (3.6.10)

« WG3:BER-060 »

3.6.14

failed outcome

<execution context> *execution outcome* (3.6.11) representing a failure caused or not recovered by the last *operation* (3.6.8) executed in the *current execution context* (3.6.10)

« WG3:BER-060 deleted one definition »

3.6.15

regular result

result that is a *value* (3.8.1) produced by the successful *execution* (3.6.7) of an *operation* (3.6.8)

3.6.16

omitted result

result indicating the successful *execution* (3.6.7) of an *operation* (3.6.8) that produced no *value* (3.8.1)

3.6.17

result type

data type (3.9.2) of a result

« WG3:BER-081 deleted two definitions »

3.6.18

current execution result

result of the *current execution outcome* (3.6.12)

Note 1 to entry: See Subclause 4.8.4, "Execution outcomes".

3.6 Procedure terms and definitions

3.6.19

evaluation

computation of a result that is not permitted to cause *side effects* (3.6.1)

« WG3:BER-081 deleted one definition »

« WG3:BER-040R3 »

3.6.20

referent

GQL-object (3.4.17) identified by a *reference value* (3.8.5)

3.6.21

catalog-modifying procedure

procedure (3.6.24) whose *execution* (3.6.7) may perform *catalog-modifying* (3.6.2) *side effects* (3.6.1) and *data-populating* (3.6.5) side effects only

3.6.22

data-modifying procedure

procedure (3.6.24) whose *execution* (3.6.7) is not permitted to perform *catalog-modifying* (3.6.2) *side effects* (3.6.1), *session-modifying* (3.6.3) side effects, or *transaction-modifying* (3.6.4) side effects

3.6.23

stored procedure

persistent (3.2.9) *procedure* (3.6.24) in the *GQL-catalog* (3.5.1)

3.6.24

procedure

description of a computation on input arguments whose *execution* (3.6.7) computes an *execution outcome* (3.6.11) and optionally causes *side effects* (3.6.1)

Note 1 to entry: See Subclause 4.10.2, "Procedures".

3.6.25

procedure signature

<procedure> *declaration* (3.7.1) of the list of mandatory *procedure* (3.6.24) parameters required, optional *procedure* (3.6.24) parameters allowed together with default values to be used when they are not given, the kinds of *side effects* (3.6.1) possibly performed, and the *procedure* (3.6.24) result type of the result returned by a complete and successful *execution* (3.6.7) of the *procedure* (3.6.24)

Note 1 to entry: See Subclause 4.10.2.1, "General description of procedures".

3.6.26

procedure logic

<procedure> operations (such as statements) together with the order in which they have to be effectively performed to completely execute the algorithm that is implemented by the *procedure* (3.6.24)

3.6.27

formal parameter

(formal parameter name, formal parameter type, parameter cardinality) triple describing an input argument of a *procedure* (3.6.24) call

3.6.28

parameter cardinality

<formal parameter> specification of the number of parameter values that may be provided for the *formal parameter* (3.6.27) by the input *binding table* (3.8.15) in a *procedure* (3.6.24) call

3.6.29

single parameter cardinality

parameter cardinality (3.6.28) of one

Note 1 to entry: A formal parameter with single parameter cardinality declares a *fixed variable* (3.8.13) in the scope of the procedure.

3.6.30

multiple parameter cardinality

parameter cardinality (3.6.28) of zero, one, or more

Note 1 to entry: A formal parameter with multiple parameter cardinality declares an *iterated variable* (3.8.14) in the scope of the procedure.

3.6.31

simple view

view (3.6.33) that expects no arguments

3.6.32

parameterized view

view (3.6.33) that expects one or more arguments

3.6.33

view

query (3.6.34) that returns a *graph* (3.3.1)

3.6.34

query

procedure (3.6.24) whose *execution* (3.6.7) may perform *data-populating* (3.6.5) *side effects* (3.6.1) only

3.6.35

function

query (3.6.34) whose *execution* (3.6.7) will not access the *GQL-catalog* (3.5.1) or the current *GQL-session* (3.4.6) in any way

« WG3:W21-010 P00-USA-377 »

« WG3:BER-086R1 »

3.6.36

command

operation (3.6.8) in a GQL-program that is not part of the *procedure* (3.6.24) in that GQL-program

Note 1 to entry: See Subclause 4.10.3, “Commands”.

3.6.37

session command

command (3.6.36) that may only perform *session-modifying* (3.6.3) *side effects* (3.6.1)

3.6.38

transaction command

command (3.6.36) that may only perform *transaction-modifying* (3.6.4) *side effects* (3.6.1)

3.6.39

working schema

GQL-schema (3.5.5) that is implicitly accessed or manipulated by the *execution* (3.6.7) of a *procedure* (3.6.24)

3.6.40

working graph

graph (3.3.1) that is implicitly accessed or manipulated by the *execution* (3.6.7) of a *procedure* (3.6.24)

3.6 Procedure terms and definitions

3.6.41

working table

binding table (3.8.15) that is implicitly transformed by the *execution* (3.6.7) of a *procedure* (3.6.24)

3.6.42

working record

record (3.8.16) that is implicitly accessed or manipulated by the *execution* (3.6.7) of a *procedure* (3.6.24)

« WG3:BER-086R1 »

3.6.43

GQL-procedure

procedure (3.6.24) written in the GQL language

3.6.44

external procedure

procedure (3.6.24) provided via an implementation-defined mechanism

3.6.45

statement

operation (3.6.8) executed as part of executing a *procedure* (3.6.24) that updates the *current execution context* (3.6.10) and its *current execution outcome* (3.6.12) and that may cause *side effects* (3.6.1)

« WG3:BER-081 deleted three definitions »

3.7 Procedure syntax terms and definitions

3.7.1

declaration

<“X” declaration> syntax that declares an “X”

3.7.2

definition

<“X” definition> syntax that defines an “X”

3.7.3

scope

<name> one or more BNF non-terminal symbols within which a name is effective

3.7.4

scope

<working object> one or more BNF non-terminal symbols within which a working object is available

3.7.5

scope clause

<name> BNF non-terminal symbol used in defining the *scope* (3.7.3) of the introduced name

3.7.6

scope clause

<working object> BNF non-terminal symbol used in defining the *scope* (3.7.4) of the declared working object

3.7.7

namespace

classification scheme that permits a given name to identify multiple objects that are distinguished by context

3.7.8**assignment**

operation whose effect is to ensure that the value at a site T (known as the target) is identical to a given value S (known as the source)

Note 1 to entry: Assignment is frequently indicated by the use of the phrase “ T is set to S ” or “the value of T is set to S ”.

3.7.9**instance**

<“X” instance> physical representation of an “X”

Note 1 to entry: Each instance is at exactly one *site* (3.7.10). An “X” instance has a type that is the type of “X”.

3.7.10**site**

place occupied by an *instance* (3.7.9) of a specified type (or subtype of that type)

3.7.11**reference**

<“X” reference> *instance* (3.7.9) that identifies a *site* (3.7.10) containing an “X”

3.7.12**referent**

<“X” referent> *instance* (3.7.9) identified by an “X” *reference* (3.7.11)

3.7.13**pattern**

syntax for abstractly specifying a collection of *pattern matches* (3.7.23)

3.7.14**pattern macro**

named template providing syntactic abstraction for a *pattern* (3.7.13)

3.7.15**pattern variable**

<“X” pattern variable> “X” *variable* (3.2.16) that is declared in an “X” *pattern* (3.7.13)

3.7.16**edge variable**

element variable (3.7.17) that is declared in an *edge pattern* (3.7.28)

« WG3:BER-031 »

Note 1 to entry: An edge variable may be bound to a list of *edges* (3.3.12).

« WG3:BER-031 »

3.7.17**element variable**

variable (3.2.16) that may be bound to a list of *graph elements* (3.3.10)

Note 1 to entry: An element variable is either a *node variable* (3.7.20) or an *edge variable* (3.7.16).

« WG3:RKE-031 »

3.7.18**primary variable**

element variable (3.7.17) that is declared in an *element pattern* (3.7.26)

3.7.19**graph pattern variable***path variable* (3.7.21), *subpath variable* (3.7.22) or *element variable* (3.7.17)**3.7.20****node variable***element variable* (3.7.17) that is declared in a *node pattern* (3.7.27)

« WG3:BER-031 »

Note 1 to entry: A node variable may be bound to a list of *nodes* (3.3.11).

« WG3:BER-031 »

3.7.21**path variable***graph pattern variable* (3.7.19) that may be bound to a path binding that is matched by a *path pattern* (3.7.25)Note 1 to entry: The extracted path of the path binding is a *path* (3.3.7).

« WG3:BER-031 »

3.7.22**subpath variable***graph pattern variable* (3.7.19) that is declared at the head of a <parenthesized path pattern expression>**3.7.23****pattern match***dictionary* (3.2.10) of *variable* (3.2.16) bindings whose entries are related according to some specified *pattern* (3.7.13)**3.7.24****graph pattern***set* (3.2.12) of one or more *path patterns* (3.7.25)**3.7.25****path pattern**pattern that matches a *path* (3.3.7)**3.7.26****element pattern***node pattern* (3.7.27) or *edge pattern* (3.7.28)**3.7.27****node pattern***path pattern* (3.7.25) that matches a single *node* (3.3.11)**3.7.28****edge pattern***path pattern* (3.7.25) that matches a single *edge* (3.3.12)**3.7.29****label expression**expression composed from *label* (3.3.19) names using disjunction, conjunction, and negation

Note 1 to entry: Disjunction, conjunction, and negation are denoted respectively by a vertical bar "|", ampersand "&" and exclamation mark "!", with parentheses for grouping.

3.7.30**linear composition**

composition of a *sequence* (3.2.13) of suboperations that forms a *composite* (3.2.2) *operation* (3.6.8) that effectively executes the suboperations in the order given by the sequence

Note 1 to entry: The linear composition of operations over binding tables corresponds to a left lateral join between those tables.

3.7.31**whitespace**

sequence (3.2.13) of one or more characters that have the Unicode property White_Space

Note 1 to entry: Whitespace is typically used to separate <non-delimiter token>s from one another, and is always permitted between two tokens in text written in the GQL language.

3.8 Value terms and definitions

3.8.1**value**

<“X” value> definite, immutable, and irreducible unit of data of some type “X”

Note 1 to entry: Values stand for themselves. The identity of a value is independent of where it occurs.

Note 2 to entry: Values are not *objects* (3.2.7).

3.8.2**comparable**, adj.

<of a pair of values> capable of being compared

Note 1 to entry: The values of data types of the same base type can be compared one with another.

3.8.3**identical**, adj.

<of a pair of values> indistinguishable, in the sense that it is impossible, by any means specified in this document, to detect any difference between them

Note 1 to entry: For the full definition, see Subclause 21.6, “Determination of identical values”.

3.8.4**distinct**, adj.

<of a pair of comparable values> capable of being distinguished within a given context

Note 1 to entry: Informally, two values are distinct if neither is null and the values are not equal. A null value and a material value are distinct. Two null values are not distinct. See Subclause 4.12.5, “Properties of distinct”, and the General Rules of Subclause 21.7, “Determination of distinct values”.

« WG3:BER-097 deleted two definitions »

3.8.5**reference value**

value (3.8.1) representing a *reference* (3.7.11) to a *GQL-object* (3.4.17)

3.8.6**material**, adj.

<value> not the *null value* (3.8.7)

3.8.7**null value**

special *value* (3.8.1) that is used to indicate the absence of other data

Note 1 to entry: The null value of Boolean type and the truth value *Unknown* are the same value. The null values of different data types are indistinguishable.

« BER-019 »

« WG3:BER-040R3 deleted one definition »

« WG3:BER-040R3 »

3.8.8

property type

<object type definition> pair comprising a name and a *value type* (3.9.17)

Note 1 to entry: A *value* (3.8.1) of a *property* (3.3.20) is a value of the property value type of its property type.

« BER-019 Two definitions deleted »

« BER-019 »

3.8.9

binding variable

dynamic (3.4.26) *variable* (3.2.16) corresponding to a field name of a *working record* (3.6.42) or a *column name* (3.9.20) of a *working table* (3.6.41)

« BER-019 »

3.8.10

graph variable

binding variable (3.8.9) whose declared type is a *graph type* (3.9.15)

« BER-019 »

3.8.11

binding table variable

binding variable (3.8.9) whose declared type is a *graph type* (3.9.15)

« BER-019 »

3.8.12

value variable

binding variable (3.8.9) whose declared type is a *value type* (3.9.17)

« BER-019 »

3.8.13

fixed variable

binding variable (3.8.9) that is bound in the current working record

« BER-019 »

3.8.14

iterated variable

binding variable (3.8.9) that is bound in the current working table

« BER-019 Definition deleted »

« WG3:BER-040R3 »

3.8.15

binding table

primary object (3.4.19) comprising a collection of zero or more records of the same *record type* (3.9.23)

Note 1 to entry: See Subclause 4.3.5, “Binding tables”, and Subclause 4.15, “Binding table types”.

** Editor's Note (number 4) **

This differs from SQL to allow representing the unit binding table and to be compatible with existing use of binding tables in property graph query languages. See Language Opportunity [GQL-219](#).

3.8.16

record

value (3.8.1) of a *record type* (3.9.23); possibly empty *set* (3.2.12) of *fields* (3.8.17)

« WG3:BER-040R3 »

3.8.17

field

<record> pair comprising a name and a value

« BER-019 Two definitions deleted »

3.8.18

byte string

an element of the byte string type

3.8.19

character string

an element of the character string type

3.9 Type terms and definitions

« BER-019 »

3.9.1

declared type

<site or a non-terminal specifying an operation> unique *data type* (3.9.2) common to every *instance* (3.7.9) that may be assigned to a given *site* (3.7.10) or that may result from *execution* (3.6.7) or *evaluation* (3.6.19) of the *operation* (3.6.8) specified by a given non-terminal

Note 1 to entry: If and only if a non-terminal has an omitted result, then it has no declared type.

3.9.2

data type

set (3.2.12) of elements with shared characteristics representing data

Note 1 to entry: A data type characterizes the sites that may be occupied by instances of its elements.

3.9.3

supertype

<“X” supertype> data type containing all elements of some data type “X”

Note 1 to entry: If T_1 is a supertype of T_2 and T_1 and T_2 are not compatible, then T_1 is a *strict supertype* of T_2 . “Compatible” is defined in Subclause 4.12.4, “Data type terminology”.

Note 2 to entry: The antonym is *subtype* (3.9.4).

3.9.4

subtype

<“X” subtype> data type comprising only elements contained in some data type “X”

Note 1 to entry: If T_2 is a subtype of T_1 and T_1 and T_2 are not compatible, then T_2 is a *strict subtype* of T_1 . “Compatible” is defined in Subclause 4.12.4, “Data type terminology”.

Note 2 to entry: The antonym is *supertype* (3.9.3).

3.9.5

base type

set of *data types* (3.9.2) with shared characteristics

3.9.6

most specific type

<of "X"> smallest data type that includes "X"

Note 1 to entry: If a data type *A* comprises fewer elements than a data type *B*, then *A* is smaller than *B*.

3.9.7

constructed, adj.

<data type> comprising *composite* (3.2.2) elements

3.9.8

user-defined, adj.

<data type> *constructed* (3.9.7) and explicitly defined by the user

3.9.9

predefined, adj.

<data type> *atomic* (3.9.10) and provided by the GQL-implementation

« WG3:BER-074 »

3.9.10

atomic, adj.

<data type> comprising only values that are not composed of values of other data types

3.9.11

material, adj.

<data type> excluding the *null value* (3.8.7)

Note 1 to entry: The antonym is *nullable* (3.9.12).

3.9.12

nullable, adj.

<data type> including the *null value* (3.8.7)

Note 1 to entry: The antonym is *material* (3.9.11).

3.9.13

any type

data type (3.9.2) comprising every element from every *object type* (3.9.14) and *value type* (3.9.17)

3.9.14

object type

data type (3.9.2) comprising *objects* (3.2.7)

3.9.15

graph type

object type (3.9.14) describing a *graph* (3.3.1) in terms of restrictions on its labels, properties, nodes, edges, and topology

Note 1 to entry: See Subclause 4.14, "Graph types".

3.9.16

binding table type

object type (3.9.14) of every *binding table* (3.8.15) of a specified *record type* (3.9.23)

Note 1 to entry: See Subclause 4.15, "Binding table types".

« BER-019 »

3.9.17

value type

data type (3.9.2) comprising *values* (3.8.1)

Note 1 to entry: See Subclause 4.16, “Value types”.

3.9.18

reference value type

value type (3.9.17) comprising *reference values* (3.8.5)

« WG3:BER-040R3 »

3.9.19

supported property value type

value type (3.9.17) supported as the type of *property* (3.3.20) values

3.9.20

column name

field (3.8.17) name of a *column* (3.9.22)

3.9.21

column type

field (3.8.17) *value type* (3.9.17) of a *column* (3.9.22)

« BER-019 »

3.9.22

column

field type (3.9.24) of the *record type* (3.9.23) of a *binding table type* (3.9.16)

« BER-019 »

3.9.23

record type

value type (3.9.17) that describes the *set* (3.2.12) of *fields* (3.8.17) of a *record* (3.8.16) in terms of their *field type* (3.9.24)

Note 1 to entry: See Subclause 4.16.1.5, “Record type”.

3.9.24

field type

pair comprising a *field* (3.8.17) name and a *field* (3.8.17) *value type* (3.9.17)

3.9.25

nothing type

empty *data type* (3.9.2)

Note 1 to entry: The nothing type has no instances.

3.10 Temporal terms and definitions

3.10.1

time

[SOURCE: ISO 8601-1:2019, 3.1.1.2]

3.10.2

instant

[SOURCE: ISO 8601-1:2019, 3.1.1.3]

**3.10.3
time scale**

[SOURCE: ISO 8601-1:2019, 3.1.1.5]

**3.10.4
duration**

[SOURCE: ISO 8601-1:2019, 3.1.1.8]

**3.10.5
time of day**

[SOURCE: ISO 8601-1:2019, 3.1.1.16]

**3.10.6
Gregorian calendar**

[SOURCE: ISO 8601-1:2019, 3.1.1.19]

**3.10.7
time shift**

[SOURCE: ISO 8601-1:2019, 3.1.1.25]

**3.10.8
calendar date**

[SOURCE: ISO 8601-1:2019, 3.1.2.7]

**3.10.9
representation with reduced precision**

[SOURCE: ISO 8601-1:2019, 3.1.3.7]

**3.10.10
negative duration**

[SOURCE: ISO 8601-2:2019, 3.1.1.7]

3.11 Definitions taken from ISO/IEC 14651:2019

**3.11.1
collation**

[SOURCE: ISO/IEC 14651:2019, 3.7]

4 Concepts

4.1 Use of terms

The concepts on which this document is based are described in terms of objects, in the usual sense of the word.

Some objects are a component of an object on which they depend. If an object ceases to exist, then every object dependent on that object also ceases to exist. The representation of an object is known as a descriptor. The descriptor of an object represents everything that is required to be known about the object. See also [Subclause 5.3.4, “Descriptors”](#).

4.2 GQL-environments and their components

4.2.1 General description of GQL-environments

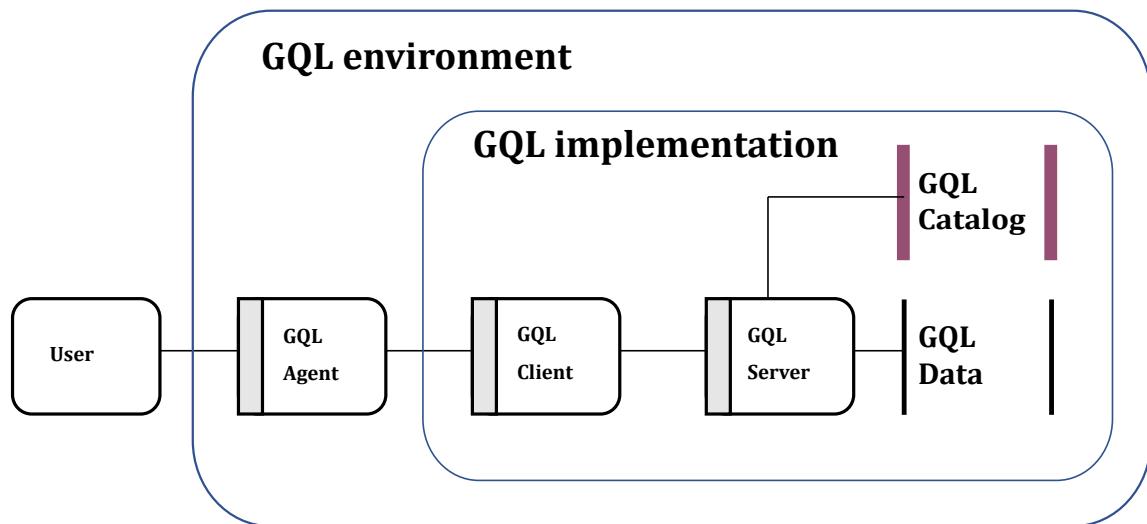


Figure 1 — Components of a GQL-environment

A *GQL-environment* is a milieu in which the *GQL-catalog* and *GQL-data* exists and *GQL-requests* are executed. A pictorial overview is shown in [Figure 1, “Components of a GQL-environment”](#). A GQL-environment comprises:

- 1) One GQL-agent.
- 2) One GQL-implementation containing one GQL-client and one GQL-server.
- 3) Zero or more authorization identifiers that identify principals.
- 4) One GQL-catalog that comprises one GQL-catalog root.
- 5) The sites, principally catalog objects, that contain GQL-data, as described by the content of the GQL-schemas. This data can be thought of as “the database”, but the term is not used in this document, because it has different meanings in the general context.

4.2 GQL-environments and their components

This document recognizes that an installation of a software component that implements GQL can provide multiple GQL-clients, multiple GQL-servers, and multiple GQL-catalogs such that some GQL-agents can use multiple GQL-clients and some GQL-servers can share the same GQL-catalog. In such a scenario, every interaction of one GQL-agent, one GQL-client, one GQL-server, and one GQL-catalog is understood to occur in an isolated GQL-environment. Each such GQL-environment is considered separately in terms of conformance. Any interaction between multiple such GQL-environments is implementation-dependent ([UA001](#)) and is understood as the activity of additional GQL-agents.

4.2.2 GQL-agents

A *GQL-agent* is that which utilizes an implementation-defined ([IW001](#)) mechanism to instruct a GQL-client to create and destroy GQL-sessions to GQL-servers, and to submit GQL-requests to GQL-servers, and thus cause the execution of procedures and commands by the GQL-implementation.

4.2.3 GQL-implementations

4.2.3.1 Introduction to GQL-implementations

A *GQL-implementation* is a processor that executes GQL-requests, as required by a GQL-agent. A GQL-implementation, as perceived by a GQL-agent, includes one GQL-client, to which that GQL-agent is bound, and one GQL-server. A GQL-implementation is able to conform to this document even if it allows more than one GQL-server to exist in a GQL-environment.

Because a GQL-implementation is specified only in terms of how it manages GQL-sessions and executes GQL-requests, the concept denotes an installed instance of some software component (database management system). This document does not distinguish between features of the GQL-implementation that are determined by the software implementor and those determined by the installer.

In this document some features are marked as “implementation-defined”. These features are permitted to differ in different implementations but each conforming implementation is required to fully specify that aspect. See [Subclause 24.5.2, “Claims of conformance for GQL-implementations”](#), for further information.

Each implementation-defined feature is assigned a code that is placed in parentheses after each occurrence of that implementation-defined feature in this document. See for example, [Syntax Rule 13](#) of [Subclause 20.2, “<value type>”](#):

“The maximum length of a character string is implementation-defined ([IL013](#)) but shall be greater than or equal to $2^{14}-1 = 16383$ characters.”

The codes assigned to implementation-defined features comprise the letter “I” followed by a letter and three digits.

The codes assigned to implementation-defined features are stable and can be depended on to remain constant.

For convenience, all the implementation-defined features in this document are collected together in the non-normative [Annex B, “Implementation-defined elements”](#), which lists those places in this document where an implementation-defined feature is referenced.

In this document some features are marked as “implementation-dependent”. These features are permitted to differ in different implementations, but that are not necessarily specified for any particular GQL-implementation. Indeed, a GQL-implementation is not required to exhibit consistent behaviour with regard to a given feature. Its behaviour may depend on aspects such as the chosen access path, which may vary over time. An application should not depend on the specific behaviour of any implementation-dependent feature.

Each implementation-dependent feature is assigned a code that is placed in parentheses after each occurrence of that implementation-dependent feature in this document. See for example, [General Rule 2\) b\) of Subclause 19.23, “<element_id function>”](#):

“Otherwise, let *GE* be the sole graph element in *LOE*. The result of <element_id function> is an implementation-dependent ([UV004](#)) value that encapsulates the identity of *GE* in the graph that contains *GE* for the duration of the currently executing GQL-request.

NOTE 145 — By implication, the value returned as the result of <element_id function> may be but is not guaranteed to be the same as the global object identifier of *GE*.

”

The codes assigned to implementation-dependent features comprise the letter “U” followed by a letter and three digits.

The codes assigned to implementation-dependent features are stable and can be depended on to remain constant.

For convenience, all the implementation-dependent features in this document are collected together in the non-normative [Annex C, “Implementation-dependent elements”](#), which lists those places in this document where an implementation-dependent feature is referenced.

This document recognizes that there is a possibility that GQL-client and GQL-server software components have been obtained from different implementors; it does not specify the method of interaction or communication between GQL-client and GQL-server.

4.2.3.2 GQL-clients

A *GQL-client* is a processor capable of establishing a connection to a [GQL-server](#) on behalf of a [GQL-agent](#) that is authenticated to represent a [principal](#). A GQL-client is perceived by the GQL-agent to be part of the GQL-implementation, that establishes and manages the sequence of GQL-sessions between itself and its GQL-server and maintains a GQL-status object and other state data relating to interactions between itself, the GQL-agent, and the GQL-server for its current GQL-session. A GQL-agent submits GQL-requests to a GQL-server via a GQL-client.

A GQL-implementation may detect the loss of the connection between the GQL-client and GQL-server during the execution of any statement. When such a connection failure is detected, an exception condition is raised: *transaction rollback — statement completion unknown (40003)*. This exception condition indicates that the results of the actions performed in the GQL-server on behalf of the GQL-client are unknown to the GQL-agent. Similarly, a GQL-implementation may detect the loss of the connection during the execution of a <commit command>. When such a connection failure is detected, an exception condition is raised: *connection exception — transaction resolution unknown (08007)*. This exception condition indicates that the GQL-implementation cannot verify whether the GQL-transaction was committed successfully, rolled back, or left active.

4.2.3.3 GQL-servers

A *GQL-server* is a processor capable of executing a [GQL-request](#) that was submitted by a [GQL-client](#) and delivering the outcome of that [execution](#) back to the GQL-client in accordance with the rules and definitions of the GQL language. A GQL-server is perceived by the GQL-agent to be part of the GQL-implementation, that manages the GQL-catalog and GQL-data.

The GQL-server of a GQL-environment:

- Manages the sequence of GQL-sessions taking place between itself and the GQL-client on behalf of the GQL-agent.
- Executes GQL-requests received from the GQL-client to completion and delivers request outcomes back to the GQL-client as required.

4.2 GQL-environments and their components

- Maintains the state of the GQL-session, including the authorization identifier, the GQL-transaction, and certain session defaults.

4.2.4 Basic Security Model

4.2.4.1 Principles

A *principal* is an implementation-defined ([ID001](#)) object that represents a user within a GQL-implementation.

A *principal* is identified by one or more *authorization identifiers*. The means of creating and destroying authorization identifiers, and their mapping to principals, is implementation-defined ([IW002](#)).

NOTE 1 — Allowing multiple authorization identifiers to map to a single principal allows a user to operate with different sets of privileges.

Every principal is associated with a home schema and a home graph. The manner in which this association is specified is implementation-defined ([ID002](#)).

The *current home schema* is the home schema of the current principal.

The *current home graph* is the home graph of the current principal.

NOTE 2 — In cases where it is not desired to provide a home schema for a principal, but where instead principals are required to explicitly select their GQL-schema at the start of a GQL-session, a GQL-implementation may use an empty schema, containing nothing and that no principal has rights to modify, as the home schema for such a principal.

4.2.4.2 Authorization identifiers

An authorization identifier identifies a principal and a set of privileges for that principal.

The set of privileges identified by an authorization identifier are implementation-defined ([ID003](#)).

4.2.5 GQL-catalog

4.2.5.1 General description of the GQL-catalog

The *GQL-catalog* is a persistent, hierarchically-organized collection of GQL-directories and GQL-schemas.

The GQL-catalog is pictorially represented in [Figure 2, “Components of a GQL-catalog”](#).

« [WG3:BER-061](#) »

The GQL-catalog comprises the GQL-catalog root that is either a GQL-directory or a GQL-schema. The name of the GQL-catalog root is the zero-length character string. If the GQL-catalog root is a GQL-directory, it is called the *root directory*. Otherwise the GQL-catalog root is a GQL-schema that is called the *root schema*.

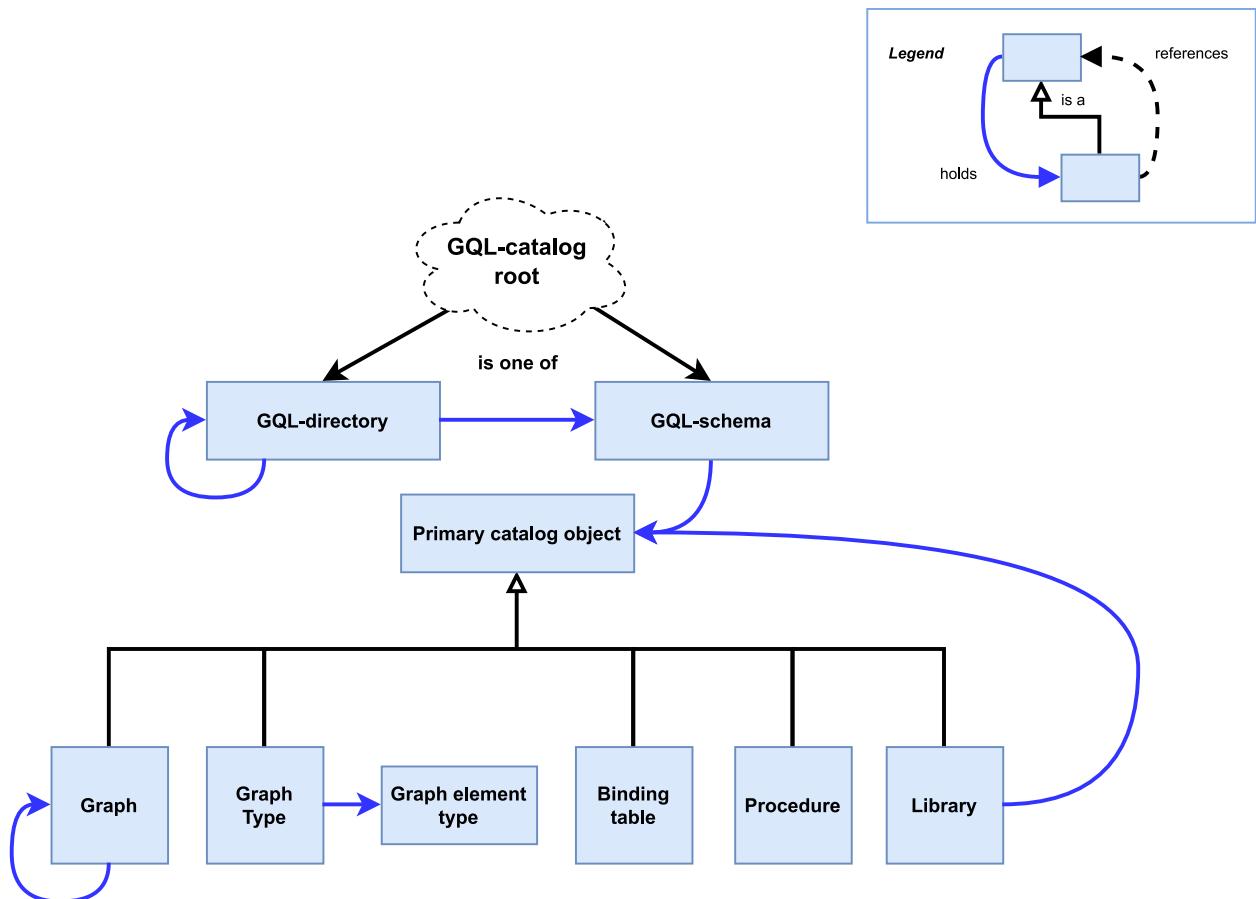


Figure 2 — Components of a GQL-catalog

« WG3:BER-061 »

The maximum depth of nesting of GQL-directories is implementation-defined (IL020). If this depth is zero, then GQL-directories are not supported.

A GQL-implementation may pre-populate the GQL-catalog with GQL-directories and GQL-schemas during its installation.

NOTE 3 — This supports GQL-implementations that provide a GQL-catalog that has only a single GQL-schema as its GQL-catalog root, has a single GQL-directory that only contains GQL-schemas as its GQL-catalog root, has a nesting structure that directly matches that of SQL such that its GQL-catalog root is a GQL-directory of SQL-catalogs (represented as GQL-directories) containing GQL-schemas, or has a nesting structure that is suitably restricted in some other way.

** Editor's Note (number 5) **

Descriptors of the GQL-catalog and objects it contains need to be defined. See Possible Problem [GQL-179](#).

4.2.5.2 GQL-directories

A *GQL-directory* is a persistent dictionary of *GQL-directories* and *GQL-schemas*.

The GQL-directory that holds some other GQL-directory or GQL-schema *DOS* is called the *parent directory* of *DOS*. The GQL-catalog root has no parent directory.

Every GQL-directory is completely described by its persistent *GQL-directory descriptor* that comprises:

4.2 GQL-environments and their components

« WG3:BER-061 »

- The name of the GQL-directory, which is either an identifier or the zero-length character string. The name of every GQL-directory *DIR* that is not the GQL-catalog root shall have a length that is greater than zero and shall uniquely identify *DIR* within its parent directory.
- The descriptors of every GQL-directory and GQL-schema that are held by the GQL-directory.

A GQL-directory shall not contain both a GQL-directory and a GQL-schema that have the equivalent name.

A GQL-implementation may automatically create a GQL-directory in an implementation-defined (IA001) way.

« WG3:BER-061 »

NOTE 4 — For instance, a GQL-implementation may (recursively) create GQL-directories when a GQL-schema is created in a thus far non-existing GQL-directory.

4.2.5.3 GQL-schemas

A *GQL-schema* is a persistent dictionary of primary catalog objects. Every GQL-schema is completely described by its persistent GQL-schema descriptor that comprises:

« WG3:BER-061 »

- The name of the GQL-schema, which is either an identifier or the zero-length character string. The name of every GQL-schema *SCHEMA* that is not the root schema shall have a length that is greater than zero and shall uniquely identify *SCHEMA* within its parent directory.
- The owner of the GQL-schema, an authorization identifier of a principal.
- The catalog object descriptors of every catalog object held by this GQL-schema.

A GQL-implementation may provide some GQL-schemas that cannot be dropped by the execution of a GQL-request.

« WG3:BER-040R3 »

A *catalog object* is a GQL-object directly or indirectly defined in the GQL-catalog. Every catalog object is created directly or indirectly in the context of a GQL-schema, and owned by that GQL-schema.

« WG3:BER-040R3 deleted one Editor's Note »

Every catalog object is described by a persistent *catalog object descriptor* that has a name that is an identifier that uniquely identifies the catalog object within its containing GQL-schema or GQL-object. A catalog object descriptor is one of:

« WG3:BER-040R3 deleted one Editor's Note »

- A named graph descriptor.
- A named graph type descriptor.
- A named procedure descriptor.
- A named library descriptor.

« WG3:BER-082R1 deleted one item »

« WG3:BER-040R3 deleted one item »

** Editor's Note (number 6) **

This document currently distinguishes between named and unnamed descriptors for some primary objects. This has not been implemented consistently though. The exact mechanism of how descriptors refer to each other and if query-local or session-local descriptors should be considered as unnamed descriptors requires further work. See Possible Problem [GQL-192](#).

NOTE 5 — A GQL-schema cannot contain a GQL-directory or a GQL-schema, because they are not catalog objects.

« WG3:BER-040R3 deleted two Notes »

A GQL-implementation may automatically populate a GQL-schema upon its creation in an implementation-defined (IA002) way.

4.2.6 GQL-data

GQL-data is data described by GQL-schemas in the GQL-catalog — data that is under the control of a GQL-implementation in a GQL-environment.

4.3 GQL-objects

4.3.1 General introduction to GQL-objects

A *GQL-object* is an object capable of being manipulated directly by the execution of a GQL-request. GQL-objects are defined by their respective content, which are either original (defined independently) or derived (defined by a computation from other objects).

« WG3:BER-040R3 »

Every GQL-object is uniquely identified by one or more effectively immutable *internal object identifiers* within the GQL-environment containing the GQL-object. A GQL-object may be associated with one such internal object identifier that is called its *global object identifier* and that reifies its identity during the execution of a GQL-request. A GQL-object associated with a global object identifier is called *globally identifiable*. The value of an object identifier is implementation-dependent (UV010) and is possibly not accessible to the user. Global object identifiers are used for definitional purposes only in order to establish the identity of GQL-objects.

Additionally, a GQL-object may have associated descriptors that define its metadata. This document defines different kinds of GQL-objects. The descriptor of a persistent *data object* describes a catalog object that has a separate existence as GQL-data. Other descriptors describe GQL-objects that have no existence distinct from their descriptors (at least as far as this document is concerned). Hence there is no loss of precision if, for example, the term “path pattern” is used when “path pattern descriptor” would be more strictly correct.

« WG3:BER-040R3 deleted one Note »

Every GQL-object is either a *primary object* or a *secondary object*.

« WG3:BER-081 »

A *primary object* is an independently definable GQL-object. Every primary object is globally identifiable and has a descriptor. Procedures or commands may define new primary objects in the GQL-catalog, assign them to *session parameters* in a GQL-session, or bind them to local variables, subject to syntax restrictions. Primary objects may also be passed as request parameters or in procedure arguments. Furthermore, primary objects may be returned as the result of an execution outcome or generally occur as the result of evaluation. Procedures and commands may interact with GQL-directories (as defined in Subclause 4.2.5.2, “GQL-directories”), GQL-schemas (as defined in Subclause 4.2.5.3, “GQL-schemas”), and the following kinds of primary objects:

« WG3:BER-040R3 deleted two items »

- Graphs, as defined in Subclause 4.3.4, “Graphs”.
- Binding tables, as defined in Subclause 4.3.5, “Binding tables”.
- Type objects, as defined in Subclause 4.3.7, “Type objects”. Subclause 4.3.n, “Type objects”.
- Procedure objects, as defined in Subclause 4.3.8, “Procedure objects”. Subclause 4.3.m, “Procedure objects”.

- Libraries, as defined in Subclause 4.3.6, “Libraries”.
- « WG3:BER-082R1 deleted one item »
- « WG3:BER-040R3 deleted one item »
- « WG3:BER-040R3 deleted one Note »

A *secondary object* is necessarily defined as a component of other GQL-objects. A procedure or command may create, modify, delete, or otherwise interact with secondary objects as long as the primary object that contains them has not been deleted. Secondary objects may be set as session parameters, passed as request parameters or in procedure parameter arguments, bound as local variables, may occur as the result of expression evaluation, or may be returned as the result of an execution outcome. Procedures and commands interact with the following kinds of secondary objects:

- Nodes.
- Edges.
- Node types.
- Edge types.
- « WG3:BER-040R3 deleted one Note »
- « WG3:BER-040R3 deleted one Editor's Note »

4.3.2 GQL-objects held by static sites

- « WG3:BER-040R3 deleted one Editor's Note »
- « WG3:BER-040R3 »

A *static site* is a **site** holding **static** instances, i.e., instances that are always determinable at request submission time. Procedures and commands may interact with the following kinds of GQL-objects held by static sites:

- « WG3:BER-040R3 deleted three items »
- Type objects as defined in Subclause 4.3.7, “Type objects”.
- « WG3:BER-040R3 deleted one Note »
- Procedures, queries, and functions, as defined Subclause 4.3.8, “Procedure objects”.
- Libraries, as defined in Subclause 4.3.6, “Libraries”.
- « WG3:BER-082R1 deleted one item »
- « WG3:BER-040R3 deleted one item »
- « WG3:BER-040R3 »

4.3.3 GQL-objects held by dynamic sites

A *dynamic site* is a **site** holding **dynamic** instances i.e., instances that are possibly only determinable at execution time. Procedures and commands may interact with the following kinds of GQL-objects held by dynamic sites:

- Graphs, as defined in Subclause 4.3.4, “Graphs”.
- Binding tables, as described by Subclause 4.3.5, “Binding tables”.
- « WG3:BER-040R3 deleted one item »
- « WG3:BER-040R3 »
- « WG3:BER-082R1 deleted one paragraph »

4.3.4 Graphs

« WG3:RKE-040 deleted one Editor's Note »

4.3.4.1 Introduction to graphs

« WG3:RKE-040 deleted one Editor's Note »

Graphs are the primary form of data that is queried and manipulated by procedures.

In the GQL language, every graph is a *property graph*. Therefore both terms can be used interchangeably, although the shorter term “graph” is preferred in this document.

A graph may be a mixed graph, a multigraph, neither, or both and is a primary object that comprises:

- A graph label set that comprises a set of zero or more labels. A label has a name that is an identifier that is unique within the graph.

**** Editor's Note (number 7) ****

It has been questioned whether the term “label set” might give the wrong impression, that any set of labels constitutes a label set, whereas it is only those sets of labels designated in the construction of a graph that qualify as label sets. Some suggestions for alternative terms are “label combination”, “label grouping” and “labeling”. See Possible Problem [GQL-232](#).

The maximum cardinality of a graph label set is implementation-defined ([IL001](#)).

Without Feature GA00, “Graph label set support”, the graph label set of a conforming graph shall be empty.

- A graph property set that comprises a set of zero or more properties. Each property comprises:
 - Its name, which is an identifier that is unique within the graph.

NOTE 6 — The names of graph labels and of graph properties are in separate namespaces.

« WG3:BER-040R3 »

- Its value, which shall be of any supported property value type.

The maximum cardinality of a graph property set is implementation-defined ([IL002](#)).

Without Feature GA01, “Graph property set support”, the graph property set of a conforming graph shall be empty.

- A set of zero or more globally identifiable nodes. Each node comprises:

« WG3:BER-040R3 deleted one item and its associated Note »

- A node label set that comprises a set of zero or more labels. A label has a name, which is an identifier that is unique within the node.

The maximum cardinality of node label sets is implementation-defined ([IL003](#)).

Without Feature GA02, “Empty node label set support”, a node in a conforming graph shall contain a non-empty node label set.

Without Feature GA03, “Singleton node label set support”, a node in a conforming graph shall not contain a singleton node label set.

Without Feature GA04, “Unbounded node label set support”, a node in a conforming graph shall not contain a node label set that comprises one or more labels.

- A node property set that comprises zero or more properties. Each property comprises:

- Its name, which is an identifier that is unique within the node.

NOTE 7 — The names of node labels and of node properties are in separate namespaces.

« WG3:BER-040R3 »

- Its value, which can be of any supported property value type.

The maximum cardinality of node property sets is implementation-defined (IL004).

- A set of zero or more globally identifiable edges. Each edge comprises:

« WG3:BER-040R3 deleted one item and its associated Note »

- An edge label set that comprises a set of zero or more labels. A label has a name, which is an identifier that is unique within the edge.

The maximum cardinality of edge label sets is implementation-defined (IL005).

Without Feature GA05, “Empty edge label set support”, an edge in a conforming graph shall contain a non-empty edge label set.

Without Feature GA06, “Singleton edge label set support”, an edge in a conforming graph shall not contain a singleton edge label set.

Without Feature GA07, “Unbounded edge label set support”, an edge in a conforming graph shall not contain an edge label set that comprises one or more labels.

- An edge property set that comprises zero or more properties. Each property comprises:

- Its name, which is an identifier that is unique within the edge.

NOTE 8 — The names of edge labels and of edge properties are in separate namespaces.

« WG3:BER-040R3 »

- Its value, which can be of any supported property value type.

The maximum cardinality of edge property sets is implementation-defined (IL006).

- Two (possibly identical) endpoints, which are nodes contained in the same graph.
- An indication whether the edge is a directed edge or an undirected edge.

Additionally, a directed edge identifies one of its endpoints as its source, and the other as its destination. The direction of a directed edge is from its source to its destination.

Without Feature GA10, “Undirected edge patterns”, a conforming graph shall not contain undirected edges.

NOTE 9 — A graph may be catalog object that has a name, but graphs may exist that have no name, or at least have no name that is required to be visible to a user of a GQL-implementation.

4.3.4.2 Graph descriptors

« WG3:RKE-040 deleted one Editor's Note »

A graph is described by a *graph descriptor* that comprises:

- The name of the graph type descriptor.
- The descriptors of every named subgraph contained in the graph.

** Editor's Note (number 8) **

The exact way in which graph type descriptors are referenced needs to be determined. A possible alternative is the use of the internal object identifier of the graph type instead of using the name. See Possible Problem [GQL-192](#).

A *named graph descriptor* is both a catalog object descriptor and a graph descriptor and comprises every component of both kinds of descriptors.

4.3.5 Binding tables

** Editor's Note (number 9) **

WG3:W05-020 suggests that this Subclause, which was accepted and adapted from [WG3:MMX-055], should be reviewed. See Possible Problem [GQL-141](#).

« BER-019 »

« WG3:BER-040R3 »

A binding table is a [primary object](#) comprising a collection of zero or more records of the same [record type](#). Binding tables mainly serve as

- primary iteration construct that drives the execution of procedures.
- a container that holds intermediary results produced by statements such as the matches found by graph pattern matching.
- a result that is returned to the GQL-agent.

Every binding table has an associated binding table descriptor that comprises:

« BER-019 »

- The binding table type of the binding table.

NOTE 10 — Binding table records may contain reference values to primary or secondary objects.

- An indication whether the binding table is declared as *ordered* or as *unordered*. If a binding table is ordered, the order of its records has been determined according to some <sort specification>.
- An indication whether the binding table is declared as *duplicate-free* or as *allowing duplicates*. A duplicate-free binding table shall not contain duplicates of any record; that is it shall not contain any two records that are not distinct.

« BER-019 »

- An optional preferred column sequence that is a permutation of the column names of the binding table type.

NOTE 11 — The preferred column sequence is metadata tracked by this document to allow implementations to provide a column sequence for a binding table that is returned as a result to the GQL-agent.

« BER-019 »

In this document, the records of the collection of records of a binding table are simply referred to as the records of the binding table. Further, the *unit binding table* is the binding table of one record whose binding table type is a unit binding table type. An *empty binding table* is a binding table of zero records.

« WG3:BER-040R3 deleted one paragraph »

« BER-019 »

The *canonical column sequence* of a binding table *BT* is defined such that if *BT* has a preferred column sequence *PCS*, then the canonical column sequence of *BT* is *PCS*. Otherwise, the canonical column sequence of *BT* is the sequence of every column name of *BT*'s binding table type in standard sort order.

In the absence of relevant additional provisions, a new binding table implicitly

- is declared as unordered;
- is declared as allowing duplicates;
- does not have a preferred column sequence.

Furthermore, if new binding table *NEW_TABLE* is obtained only by selecting a subset of the records of a given binding table *TABLE* that is explicitly declared as duplicate-free, then *NEW_TABLE* is implicitly declared as duplicate-free.

If a General Rule refers to the *i*-th record *RECORD* of an ordered binding table *TABLE*, then *RECORD* is the *i*-th record in the sequence of all records of *TABLE* in the order determined by *TABLE*.

If a General Rule *RULE* refers to the *i*-th record *RECORD* of an unordered binding table *TABLE*, then *RECORD* is the *i*-th record in some sequence of all records of *TABLE* in an implementation-dependent (US001) order determined for each application of *RULE* and any of its subrules.

If the application of a General Rule *RULE* processes all records of a binding table using order-independent terms, then the records are to be processed effectively in a sequential, implementation-dependent (US001) order determined for that application of *RULE* and any of its subrules.

« WG3:BER-040R3 »

The collection of records comprising a binding table is not modified after the initial construction and population of that binding table. The binding table assigned to a site *S* (such as the current working table) may be replaced by assigning a new binding table to *S* but is never modified after its initial construction.

« WG3:BER-040R3 deleted one paragraph and its associated list »

« WG3:BER-040R3 »

If the set of field or column names of a record or, respectively, a binding table *A* and the set of field or column names of another record or, respectively, a binding table *B* are disjoint, then *A* and *B* are called field-name compatible.

The *Cartesian product* between a record *R* and a binding table *T* that are field-name compatible is a new binding table *NT* comprising a collection of new records obtained by constructing a new record *NR* for every record *TR* of *T* such that the fields of *NR* are the fields of both *TR* and *R*. If *T* is declared as duplicate-free, then *NT* is declared as duplicate-free; otherwise, *NT* is declared as allowing duplicates.

Furthermore, the Cartesian product between two binding tables *T1* and *T2* that are field-name compatible is a new binding table *T3* comprising a collection of every record from the Cartesian products between each record of *T1* with *T2*. If both *T1* and *T2* are declared as duplicate-free, then *T3* is declared as duplicate-free; otherwise *T3* is declared as allowing duplicates.

4.3.6 Libraries

**** Editor's Note (number 10) ****

WG3:W05-020 suggests that this Subclause, which was accepted and adapted from [WG3:MMX-055], should be reviewed. See Possible Problem **GQL-142**.

« WG3:BER-040R3 »

« WG3:BER-082R1 »

A *library* is a **primary catalog object** comprising a **dictionary** of catalog objects. Libraries group catalog objects and provide qualified names for them. In this document, libraries are assumed to contain sublibraries, procedures, or constant objects only.

A library may have a default procedure that is invoked when the library is invoked as if it was a procedure.

Every library is completely described by its named library descriptor. Every named library descriptor is a catalog object descriptor. In addition to the components of a catalog object descriptor, a named library descriptor comprises:

- The optional *library default procedure name* that is the name of a procedure that is contained in the library and that is called the *library procedure object*.
 - The catalog object descriptors of every catalog object contained in the library.
- « WG3:BER-082R1 deleted one Subclause »
« WG3:BER-040R3 deleted one Subclause »
« WG3:BER-040R3 »

4.3.7 Type objects

A *type object* is a primary object that represents a user-defined or implementation-defined object type such as a graph type. Type objects are described by the descriptors of the data type that they represent.

NOTE 12 — See Subclause 4.12, "Data types" for further details.

4.3.8 Procedure objects

A *procedure object* is a primary object that represents a user-defined or implementation-defined procedure or query.

NOTE 13 — See Subclause 4.10.2, "Procedures" for further details.

4.4 Values

« WG3:BER-040R3 deleted one Editor's Note »

4.4.1 General information about values

« WG3:BER-040R3 »

In a general sense, values are definite objects. In GQL, values are used as:

- Characteristics of GQL-objects and their descriptors.
- Session parameters.
- Request parameters.
- Procedure arguments.
- Binding variables.
- Property values.
- Result values.

Values are implicitly copied by assigning them to a site or by passing them as parameters.

« WG3:BER-040R3 »

4.4.2 Property values and supported property value types

A property value is a value of a property of a GQL-object.

Every property value is the value of a supported property value type. The values of supported property value types shall not be reference values or values that contain reference values. The following (nullable) data types are supported property value types:

4.4 Values

- The variable-length character string type specified by STRING or VARCHAR.
- The Boolean type specified by BOOLEAN or BOOL.
- The signed regular integer type specified by SIGNED INTEGER, INTEGER, or INT.

Implementations may extend the set of supported property values types with additional direct value types. The set of such additionally supported property value types is implementation-defined (ID008).

« WG3:BER-040R3 deleted one paragraph »

4.4.3 Reference values

« WG3:BER-040R3 »

A *reference value* effectively encapsulates the global object identifier of the (single) globally identifiable GQL-object that it represents (its referent).

Two reference values are equal if and only if they refer to the same referent. Additional rules regarding the comparison of reference values that have different referents but are of the same base type are implementation-defined (IA018).

« WG3:BER-097 deleted one Subclause »

« WG3:BER-040R3 »

4.4.4 Values classified according to their use of references

The following are the broad classes of values according to their use of references:

- Direct values do not include indirect values.
- Indirect values are reference values or include other indirect values.

« WG3:BER-040R3 deleted one Subclause »

4.5 GQL-sessions

4.5.1 General description of GQL-sessions

A *GQL-session* is an implementation-defined (ID006) time period in which consecutive *GQL-requests* are executed by a *GQL-client* on behalf of a *GQL-agent* with the same *GQL-environment*. At any one time during a GQL-session, exactly one of these consecutive GQL-requests is being executed and is said to be an *executing GQL-request*.

A GQL-session is created either explicitly by the GQL-client, on behalf of a GQL-agent or implicitly whenever a GQL-client, on behalf of a GQL-agent, initiates a request to a GQL-server and no GQL-session is current. The GQL-session is terminated following the last request from that GQL-client, on behalf of that GQL-agent. The mechanism and rules by which a GQL-implementation determines when the last request has been received are implementation-defined (IW003).

The context of a GQL-session is manipulated by session management commands.

An executing GQL-request *ER* causes a nested sequence of consecutive (inner) procedures and commands to be executed as a direct result of *ER*; during that time, exactly one of these is also an executing procedure or an executing command and it in turn may similarly involve execution of a further nested sequence, and so on, indefinitely. An executing procedure or command *EPC* such that no procedure or command is executing as a direct result of *EPC* is called the *innermost executing procedure or command* of the GQL-session. An executing procedure *EP* such that no procedure is executing as a direct result of *EP* is called the *innermost executing procedure* of the GQL-session. An executing command *EC* such that no command is executing as a direct result of *EC* is called the *innermost executing command* of the GQL-session.

« WG3:BER-097 »

Executing each such individual procedure or command *EPC* causes a nested sequence of consecutive (sub)operations such as statements to be executed as part of executing *EPC*; during that time, exactly one of these is also an executing operation and it in turn may similarly involve execution of further, arbitrarily nested, sequences of other operations including procedures. An executing statement *ES* such that no statement is executing as a direct result of *ES* within the same innermost executing procedure or command is called the *innermost executing statement* of the GQL-session. An executing operation *EO* such that no operation is executing as a direct result of *EO* within the same innermost executing procedure or command is called the *innermost executing operation* of the GQL-session.

4.5.2 Session contexts

4.5.2.1 Introduction to session contexts

A *session context* is a context associated with a **GQL-session** in which multiple **GQL-requests** are executed consecutively. A session context comprises the following characteristics:

- The authorization identifier.
- The principal identified by the authorization identifier.
- The time zone identifier.
- The session schema that is a GQL-schema.
- The session graph that is a graph.
- The (possibly empty) dictionary of session parameters.
- The current transaction that is an optional GQL-transaction.
- The request context that is an optional GQL-request context.
- The termination flag that is a Boolean value.

NOTE 14 — The termination flag is initially set to *False* but may be set to *True* during the execution of a GQL-program by the <session close command> to signal that the current session is to be terminated.

The *current session context* is the **session context** associated with the **GQL-session** of the currently executing **GQL-request**.

« WG3:BER-010 P00-USA-357 »

As a principle, certain phrases of the form *current x* are used to refer to the corresponding GQL-*x*- or *x*-characteristic of the current session context. Concretely, exactly the following non-ambiguous forms are used in this document:

- The *current authorization identifier* is the authorization identifier of the current session context.
- The *current principal* is the principal of the current session context.
- The *current time zone identifier* is the time zone identifier of the current session context.
- The *current session schema* is the session schema of the current session context.
- The *current session graph* is the session graph of the current session context.
- The *current session parameters* are the session parameters of the current session context.
- The *current session parameter flags* are the session parameter flags of the current session context.
- The *current transaction* is the GQL-transaction of the current session context.
- The *current request context* is the request context of the current session context.

- The *current termination flag* is the termination flag of the current session context.

4.5.2.2 Session context creation

A new session context is created and initialized by setting each of its characteristics explicitly.

4.5.2.3 Session context modification

The characteristics of a session context shall only be modified after their initialization as follows:

- Setting the time zone identifier.
- Setting the session schema.
- Setting the session graph.
- Setting the session parameters.
- Setting the current transaction to the currently active GQL-transaction associated with the GQL-session immediately after that GQL-transaction is initialized.
- Setting the current transaction to no transaction immediately after the currently active GQL-transaction associated with the GQL-session is terminated.
- Setting the request context upon starting or terminating the execution of a GQL-request.
- Setting the termination flag to signal that the GQL-session is about to be terminated.

4.6 GQL-transactions

4.6.1 General description of GQL-transactions

A *GQL-transaction* (transaction) is a sequence of executions of procedures that is atomic with respect to recovery. That is to say: either the complete execution of each procedure results in a successful outcome, or there is no effect on any objects in the GQL-catalog or on GQL-data.

At any time, there is at most one currently active transaction between the GQL-agent and the GQL-server.

For the purposes of this specification there is only one currently active transaction, which is the behavior of a system that only supports serializable transactions.

Statements within a procedure effectively execute serially. The state of GQL-data and the GQL-catalog, as affected by a successfully-completed statement, are visible to a successor statement.

Any relaxation of the assumption of the serializable transactional behavior (for example, less restrictive isolation levels) is an implementation-defined ([ID007](#)) extension.

A GQL-implementation may define different kinds of GQL-transactions to be initiated for the execution of a catalog-modifying procedure, a data-modifying procedure, and a query.

« WG3:BER-022 »

If a GQL-implementation supports Feature GC01, “Catalog and data statement mixing”, a GQL-transaction can contain the execution of both data-modifying and catalog-modifying procedures. Without Feature GC01, “Catalog and data statement mixing”, executing data-modifying and catalog-modifying procedures in the same GQL-transaction results in an exception. If permitted, there may be additional implementation-defined ([ID009](#)) restrictions, requirements, and conditions. If any such restrictions, requirements, or conditions are violated, then an exception condition or a completion condition warning is raised with an implementation-defined ([ID010](#)) class and/or subclass code.

4.6.2 Transaction demarcation

Starting and terminating of transactions may be accomplished by implementation-defined ([IW004](#)) means (including proprietary agent-server protocols, transaction managers that act out of band to agent-server interactions, auto-starting, and auto-rollback on exception of transactions) but any GQL-implementation shall make available the following explicit transaction demarcation commands, to be issued by agents in GQL-requests to a GQL-server.

A GQL-transaction can be initiated by a GQL-agent submitting a GQL-request that specifies a <start transaction command>. If no GQL-transaction has been initiated, the commencement of execution of a procedure will initiate a GQL-transaction.

The <start transaction command> allows the specification of the characteristics of the GQL-transaction, such as the mode (read-only or read-write). If a <start transaction command> is submitted when there is already an active GQL-transaction, then an exception condition is raised: *invalid transaction state — active GQL-transaction* ([25G01](#)).

Every GQL-transaction is terminated by an attempt to either commit or a rollback. A successful rollback causes the transaction to have no effect on the GQL-catalog or GQL-data; a successful commit causes the execution outcome to be completely successful.

The GQL-agent that initiated a transaction may request the GQL-implementation to either commit or rollback that transaction by submitting a GQL-request that specifies a transaction commit command or a transaction rollback command to the GQL-server.

The transaction demarcation commands may be submitted in the same GQL-request as a procedure. A GQL-request may state a <start transaction command> prior to the procedure, allowing specifying characteristics of the transaction started for that procedure. A GQL-request may state a transaction commit command or a transaction rollback command following the procedure. A GQL-request containing a transaction commit command following a procedure causes a request to the GQL-server to terminate the currently active GQL-transaction by committing upon the successful completion of that procedure. A GQL-request containing a transaction rollback command following a procedure causes a request to the GQL-server to terminate the currently active GQL-transaction by rolling back upon the successful completion of that procedure.

If a GQL-transaction becomes blocked, cannot complete without causing semantic inconsistency, or if the resources required to continue its execution become unavailable, then a GQL-implementation may force a rollback.

Any failure within a GQL-request procedure will cause an attempt by the GQL-implementation to rollback the current transaction.

**** Editor's Note (number 11) ****

It would be preferable to (be able to) decide in the application logical whether to commit or rollback. See [Language Opportunity GQL-030](#).

Once a commit or rollback has been issued, then no further statements will be executed in the sequence of statements that make up the transaction.

Once requested, transaction termination shall succeed or fail. If a termination request cannot be processed successfully, then the state of the GQL-catalog and GQL-data becomes indeterminate. The ways in which termination success or failure statuses are made available to the GQL-agent or to an administrator is implementation-defined ([IW005](#)).

4.6.3 Transaction isolation

Provisional (uncommitted) changes to the GQL-catalog or GQL-data state that are made in the context of a transaction executing a catalog-modifying procedure or a data-modifying procedure may be visible, at

some point or to some degree, to other GQL-agents executing a concurrent transaction. The levels of transaction isolation, their interactions, their granularity of application, and the means of selecting them are implementation-defined (ID011).

NOTE 15 — GQL-implementations are therefore free to use the isolation levels defined in SQL, or more recently-defined levels such as Snapshot Isolation or Serializable Snapshot Isolation, or other industrially-applied or theoretical variants.

4.6.4 Encompassing transaction belonging to an external agent

In some environments (e.g., remote database access), a GQL-transaction can be part of an encompassing transaction that is controlled by an agent other than the GQL-agent.

In such environments, an encompassing transaction shall be terminated via that other agent, which in turn interacts with the GQL-implementation via an interface that may be different from GQL (COMMIT or ROLLBACK), in order to coordinate the orderly termination of the encompassing transaction. If the encompassed GQL-transaction is terminated by an implicitly initiated <rollback command>, then the GQL-implementation will interact with that other agent to terminate that encompassing transaction. The specification of the interface between such agents and the GQL-implementation is beyond the scope of this document. However, it is important to note that the semantics of a GQL-transaction remain as defined in the following sense:

- When an agent that is different from the GQL-agent requests the GQL-implementation to rollback a GQL-transaction, the General Rules of Subclause 8.4, “<rollback command>”, are performed.
- When such an agent requests the GQL-implementation to commit a GQL-transaction, the General Rules of Subclause 8.5, “<commit command>”, are performed. To guarantee orderly termination of the encompassing transaction, this commit operation may be processed in several phases not visible to the application; it is not required that all of the General Rules of Subclause 8.5, “<commit command>”, are executed in a single phase.

4.7 GQL-requests

4.7.1 General description of GQL-requests

« WG3:BER-086R1 »

A GQL-request is the basic unit of communication between a GQL-client and a GQL-server.

A GQL-request comprises:

- The request source that is a GQL-program.
- The (possibly empty) dictionary of request parameters that is determined using implementation-defined (IW006) means.

« WG3:BER-086R1 »

A GQL-program consists of commands and a GQL-procedure; the GQL-procedure in turn consists of statements. The request outcome of a GQL-request is the execution outcome of the requested GQL-program.

4.7.2 GQL-request contexts

4.7.2.1 Introduction to GQL-request contexts

A *GQL-request context* is a context that augments a *session context* in which an individual *GQL-request* is executed and that comprises the following characteristics:

« WG3:BER-076 deleted one list item »

- The (possibly empty) dictionary of request parameters.

- The execution stack that is a push-down stack of execution contexts.
- The request outcome that is an execution outcome.

« WG3:BER-076 deleted three paragraphs »

The *current request context* is a GQL-request context associated with the currently executing GQL-request.
« WG3:BER-010 P00-USA-357 »

As a principle, certain phrases of the form *current x* are used to refer to the corresponding *x*-characteristic of the current request context. Concretely, exactly the following non-ambiguous forms are used in this document:

« WG3:BER-076 deleted one list item »

- The *current request parameters* are the request parameters of the current request context.
- The *current execution stack* is the execution stack of the current request context.
- The *current request outcome* is the request outcome of the current request context.

4.7.2.2 GQL-request context creation

A new GQL-request context is created and initialized implicitly before the application of the Syntax Rules of <GQL-request> by determining the (possibly empty) dictionary of request parameters using implementation-defined (IW006) means. The execution stack and the request outcome are set explicitly by the General Rules of <GQL-request>.

4.7.2.3 GQL-request context modification

« WG3:BER-076 deleted one paragraph »

« WG3:BER-076 »

In this document, evaluation of General Rules only modifies an already initialized GQL-request context by setting its request outcome.

4.7.3 Working objects

The execution of procedures, statements, and commands interacts with GQL-objects which are available as a *working object*. A working object of kind *X* is an existing object of some kind *X*. A working object has a scope and is declared by a BNF non-terminal *NT* and made available as a working object of kind *X* to all elements in its scope.

« Editorial: Fix wording »

NOTE 16 — In this document, the scope of a working object is determined by the BNF non-terminal that declares it.

For a BNF element *E*, the working object of some kind *X* that is declared by the BNF non-terminal with the innermost scope clause containing *E* and whose scope includes *E* is said to be the innermost working object of kind *X* for *E*. If a working object *CWO_x* of some kind *E* is available at a BNF element *E*, then *CWO_x* is uniquely determined to be the innermost working object of kind *X* for *E*. Consequently, in the context of a given BNF non-terminal, zero or exactly one working object of each kind is available.

GQL-requests declare the following working objects:

- A *working schema* that is a GQL-schema. A GQL-schema *S* is declared by a BNF non-terminal *BNF*, such as a <GQL-request> or an <at schema clause>. *BNF* is said to declare *S* as a working schema.
- A *working graph* is a graph. A graph *G* is declared by a BNF non-terminal *BNF*, such as a <GQL-request> or <use graph clause>. *BNF* is said to declare *G* as a working graph.

As a principle, phrases of the form *current X* of *A* are used to refer to the uniquely determined working object of kind *X* that is available at BNF non-terminal *A*. If no working object of kind *X* is available at *A*, an exception condition is raised: *syntax error or access rule violation — invalid syntax (42001)*. Concretely, the following non-ambiguous phrase forms are used in this document to refer to working objects:

- The *current working schema of A* is the working schema that is available at *A*.
- The *current working graph of A* is the working graph that is available at *A*.

Additionally, phrases of the form *current X* are used to refer to the working *X* of *A* such that *A* is the BNF non-terminal of the syntactic element defined by the subclause in which the phrase is used.

4.7.4 Execution stack

The *execution stack* is a push-down stack of *execution contexts* associated with a *GQL-request*. There is one cell on this stack that is initialized with a new execution context when execution of the GQL-request begins. This execution context is always the lowest execution context in the execution stack.

An additional execution context is pushed on the stack for each procedure or command that is executed, and is removed when that procedure or command completes execution. An additional child execution context may be pushed on the stack for the execution of a command or a statement, the evaluation of an expression, or the processing of some other non-terminal that specifies an operation to be executed, and if such an execution context has been created, it is removed when that execution, evaluation, or processing completes.

NOTE 17 — Changes to the execution stack are always explicitly specified by General Rules.

The highest execution context in the execution stack of the currently executing GQL-request is called the *current execution context*.

4.8 Execution contexts

4.8.1 General description of execution contexts

An *execution context* is a context comprising a *dictionary* of objects that is associated with and manipulated by the *execution of operations*. It provides access to visible objects, allowing their manipulation by the execution of a procedure or a command.

An execution context comprises the following characteristics:

- The working table that is a binding table.
- The working record that is a record.
- The execution outcome that is an execution outcome.

The atomicity of multiple operations that are executed in an execution context depends on the types of the operations.

The *current execution context* is an *execution context* of the currently executing *operation* of the currently executing *procedure or command*.

« WG3:BER-010 P00-USA-357 »

As a principle, certain phrases of the form *current x* are used to refer to the corresponding *x*-characteristic of the current execution context. Concretely, exactly the following non-ambiguous forms are used in this document:

- The *current working table* is the working table of the current execution context.
- The *current working record* is the working record of the current execution context.

- The *current execution outcome* is the execution outcome of the current execution context.
- The *current execution result* is the result of the current execution outcome.

« BER-019 »

In this document, the Syntax Rules of a BNF non-terminal *X* that is contained in <GQL-request> can refer to the following additional transient sites:

- The *incoming working table* of *X* is the current working table immediately before the application of the first General Rule of *X*.
- The *incoming working record* of *X* is the current working record immediately before the application of the first General Rule of *X*.
- The *outgoing working table* of *X* is the current working table immediately after the application of the last General Rule of *X*.
- The *outgoing working record* of *X* is the current working record immediately after the application of the last General Rule of *X*.

If a BNF non-terminal instance *B* is immediately contained in a BNF non-terminal instance *A*, then the following conditions are implicit, unless explicitly specified otherwise:

- The declared type of the incoming working table of *B* is the declared type of the incoming working table of *A*.
- The declared type of the incoming working record of *B* is the declared type of the incoming working record of *A*.

If a BNF non-terminal instance *B* is the last instance immediately contained in a BNF non-terminal instance *A*, then the following conditions are implicit, unless explicitly specified otherwise:

- The declared type of the outgoing working table of *A* is the declared type of the outgoing working table of *B*.
- The declared type of the outgoing working record of *A* is the declared type of the outgoing working record of *B*.
- The declared type of *A* is the declared type of *B*.

4.8.2 Execution context creation

A new execution context is be created by one of the following approaches:

« WG3:BER-051R1 »

- Creating a new execution context that is initialized by setting each of its characteristics explicitly.
- Creating a new *child execution context* that is initialized by:
 - Copying each of its characteristics from the corresponding characteristics of the current execution context.
 - Setting its working table to a unit binding table.

« WG3:BER-051R1 »

The creation of a new child execution context may be further modified by overriding some of its characteristics using a phrase such as “with ... as its *CHARACTERISTIC*” or similar grammatical variants and by resetting some of its characteristics to implementation-defined defaults using a phrase such as “with the default *CHARACTERISTIC*” or similar grammatical variants.

4.8 Execution contexts

The execution of an *ACTION* in a new child execution context using a phrase such as “perform *ACTION* in a new child execution context” or similar grammatical variants is defined as follows:

« WG3:BER-051R1 »

- 1) Create a new child execution context *CTX*.
 - 2) Push *CTX* on the current execution stack.
 - 3) Perform *ACTION*.
- « WG3:BER-051R1 deleted two items »
- 4) Pop the current execution stack.

NOTE 18 — A GQL-implementation may choose to implement an instruction to execute an *ACTION* in a new child execution context by instead executing it directly in the current execution context as long as that direct execution is indistinguishable from the execution in a new child execution context that was specified by the instruction.

« WG3:BER-051R1 deleted 2 paragraphs »

« WG3:BER-051R1 »

The execution of an *ACTION2* in a new child execution context amended with some record *R* using a phrase such as “perform *ACTION2* in a new child execution context amended with *R*” or similar grammatical variants is defined as follows:

- 1) Let *AWR* be a new record that comprises:
 - a) Every field of the original current working record whose field name differs from the field name of any field in *R*.
 - b) Every field of *R*.

NOTE 19 — The definition gives precedence to a field *FR* in *R* over a field *FO* in the original current working record if *FR* and *FO* have the same field name. This document generally aims to prevent this case from arising by other means such as syntax restrictions on variable name declarations.

- 2) Let *ACTION1* comprise:
 - a) Set the current working record *AWR*.
 - b) Perform *ACTION2*.
- 3) Perform *ACTION1* in a new child execution context.

4.8.3 Execution context modification

« WG3:BER-062 »

In this document an execution context is only modified after its initialization as follows:

- Setting the working table.
- Setting the working record.
- Setting the execution outcome.

« WG3:BER-062 deleted one paragraph »

4.8.4 Execution outcomes

An *execution outcome* is a component of an *execution context* representing the outcome of an *execution* and comprises:

« WG3:BER-099R1 »

- A *status* that is always a GQL-status object.

« WG3:BER-060 »

- An optional *result* that is always a value.

NOTE 20 — Dynamically constructed objects are represented by reference values that identify them.

An execution outcome is one of:

« WG3:BER-060 »

- A *successful outcome* that is an **execution outcome** representing the successful and complete **execution** of the last **operation** executed in the **current execution context**. The result of a successful outcome is one of:
 - A *regular result* is a result that is a **value** produced by the successful **execution** of an **operation**.
 - An *omitted result* is a result indicating the successful **execution** of an **operation** that produced no **value**.

« WG3:BER-060 »

- A *failed outcome* that is an **execution outcome** representing a failure caused or not recovered by the last **operation** executed in the **current execution context**. The failed outcome has no result.

The GQL-status object of a successful outcome has a GQLSTATUS that corresponds to a completion condition and that indicates that any previous errors have been recovered and no new error was raised by the last operation executed in the execution context.

The GQL-status object of a failed outcome has a GQLSTATUS that corresponds to an exception condition and that indicates that a previously raised error was not recovered or that a new error was raised by the last operation executed in the execution context.

« WG3:BER-099R1 »

If a non-terminal *NT* has an outcome, then it is an execution outcome and the result and status of *NT* are the result and status, respectively, of the outcome of *NT*.

4.9 Diagnostics

4.9.1 Introduction to diagnostics

Every GQL-program returns diagnostic information to the GQL-client that originated the GQL-request of which the GQL-program was a part.

This diagnostic information is contained in a GQL-status object that minimally comprises a condition code but may also include additional diagnostic information.

It is implementation-defined ([IW007](#)) how a GQL-status object is presented to a GQL-client.

NOTE 21 — For example, in a Java environment a GQL-status object with a GQLSTATUS whose corresponding condition's category is "X" may cause an exception to be thrown.

4.9.2 Conditions

A condition is represented by the GQLSTATUS character string to which it corresponds. The format of GQLSTATUS character strings is specified in Subclause 23.1, "GQLSTATUS".

There are two types of conditions:

- completion conditions
- exception conditions

4.9 Diagnostics

A *completion condition* is one that permits a statement to have an effect other than that associated with raising the condition. The completion conditions comprise the conditions: *successful completion (00000)*, *warning (01000)*, and *no data (02000)*, including all subclass code variants.

The completion condition *warning (01000)* is broadly defined as completion in which the effects are correct, but there is reason to caution the user about those effects. The subclass code provides information as to the specific reason for the warning. It is raised for implementation-defined ([ID015](#)) conditions as well as conditions specified in this document. The completion condition *no data (02000)* has special significance and is used to indicate an empty result. The completion condition *successful completion (00000)* is defined to indicate a completion condition that does not correspond to *warning (01000)* or *no data (02000)*.

If no other completion or exception condition has been specified, then completion condition *successful completion (00000)* is returned. This includes conditions represented by a GQLSTATUS whose subclass code provides implementation-defined information of a non-cautionary nature.

An *exception condition* is one that causes a statement to have no effect other than that associated with raising the condition (that is, not a completion condition).

If a GQL-request does not conform to the Format, Syntax Rules, and Conformance Rules of <GQL-request>, then an exception condition is raised. Unless a Syntax Rule is violated that specifies the explicit exception condition *syntax error or access rule violation — invalid reference (42002)*, an exception condition is raised: *syntax error or access rule violation — invalid syntax (42001)*.

Except where otherwise specified, the phrase “an exception condition is raised:”, followed by the name of a condition, is used in General Rules and elsewhere to indicate that:

- The execution of a statement is unsuccessful.
- The application of the General Rules is terminated, unless explicitly stated otherwise.
- Diagnostic information is made available.
- Execution of the statement is to have no effect on GQL-data or the GQL-catalog.

The phrase “a completion condition is raised:”, followed by the name of a condition, is used in General Rules and elsewhere to indicate that application of General Rules is not terminated and diagnostic information is made available; unless an exception condition is also raised, the execution of the statement is successful.

If more than one condition could have occurred as a result of a statement, it is implementation-dependent ([UA002](#)) whether diagnostic information pertaining to more than one condition is made available. Those additional conditions, if any, are placed in a separate GQL-status object and chained to the GQL-status object containing the GQLSTATUS with the greatest precedence.

For the purpose of choosing the GQLSTATUS to be returned:

- Every exception condition for transaction rollback has precedence over every other exception condition.
- Every exception condition has precedence over every completion condition.
- The completion condition *no data (02000)* has precedence over the completion condition *warning (01000)*.
- The completion condition *warning (01000)* has precedence over the completion condition *successful completion (00000)*.

The values assigned to GQLSTATUS shall obey these precedence requirements.

4.9.3 GQL-status object

A *GQL-status object* is one in which one or more *conditions* are recorded as they arise.

« WG3:BER-072 »

Whenever a statement is executed, it sets values, representing one or more conditions resulting from that execution, in a GQL-status object. These values give some indication of what has happened. The diagnostic information is returned as part of the request outcome to the GQL-client.

Each GQL-status object comprises:

- A GQLSTATUS that is a GQLSTATUS character string.
« Editorial: Eliminate mixed lists »
- A GQLSTATUS description that is a character string describing the GQLSTATUS.
- A map with diagnostic information as defined in Clause 22, “Diagnostics”.
- A (possibly empty) chain of nested GQL-status objects.
- A map of implementation-defined (ID017) diagnostic information.

NOTE 22 — For example, this may include a stack trace.

« Editorial: Eliminate mixed lists »

A GQLSTATUS description is either a standard description or an implementation-defined (ID016) translation of that standard description appropriate for the locale of the GQL-client.

A *standard description* is defined as follows:

- 1) Let *CAT*, *SUBCLASS*, *COND*, and *SUBCOND* be the Category, Subclass, Condition, and Subcondition fields of the associated row in Table 7, “GQLSTATUS class and subclass codes”, for the condition corresponding to the GQLSTATUS.
- 2) Let *TEXT* be determined as follows: If *SUBCLASS* is '000', then *TEXT* is *COND*; otherwise, *TEXT* is the concatenation of *COND*, '-' , and *SUBCOND*.
- 3) Case:
 - a) If *CAT* is 'X', then the standard description is the concatenation of 'error: ' and *TEXT*.
 - b) If *CAT* is 'W', then the standard description is the concatenation of 'warning: ' and *TEXT*.
 - c) Otherwise, the standard description is the concatenation of 'note: ' and *TEXT*.

At the beginning of any execution, the current GQL-status object is emptied. A GQL-implementation places diagnostic information about a completion condition or an exception condition corresponding to the GQLSTATUS into this GQL-status object. If other conditions are raised, the extent to which further GQL-status objects are chained is implementation-defined (ID018).

4.10 Procedures and commands

« WG3:BER-086R1 deleted one Editor's Note »

4.10.1 General description of procedures and commands

« WG3:BER-086R1 »

Procedures and commands are executed as part of executing a GQL-program in order to:

- Read or modify the GQL-catalog and the catalog objects it contains.

4.10 Procedures and commands

- Read or modify GQL-data.
- Read or modify session context characteristics such as their session graph, session schema, session parameters, or GQL-transactions.
- Read request parameters.
- Read, modify, or manipulate GQL-objects that are reachable via the current execution context, the current request context, or the current session context.

4.10.2 Procedures

**** Editor's Note (number 12) ****

Bindings for host languages should eventually be defined. See Language Opportunity [GQL-003](#).

4.10.2.1 General description of procedures

A procedure is a description of a computation on input arguments whose [execution](#) computes an [execution outcome](#) and optionally causes [side effects](#). In this document, a procedure is represented by a primary object that defines the *procedure logic* of the procedure; the procedure logic is operations (such as statements) together with the order in which they have to be effectively performed to completely execute the algorithm that is implemented by the [procedure](#).

These operations may read, modify, or otherwise manipulate the GQL-catalog, the GQL-data, the GQL-session, or other objects reachable via the current execution context, the current request context, or the current session context that are available during their execution.

4.10.2.2 Procedure descriptors

A procedure is described by a *procedure descriptor* that comprises:

- A <schema reference> to the working schema of the procedure.
- A <catalog graph reference> to the working graph of the procedure.
- The procedure signature descriptor of the procedure.
- Zero or more of the types of side effects described in [Subclause 4.10.5.3, “Operations classified by type of side effects”](#), that may be performed when the procedure is executed.

A *named procedure descriptor* is both a catalog object descriptor and a procedure descriptor and comprises every component of both kinds of descriptors.

4.10.2.3 Procedure signatures

The *procedure signature* of a procedure is a [declaration](#) of the list of mandatory [procedure](#) parameters required, optional [procedure](#) parameters allowed together with default values to be used when they are not given, the kinds of [side effects](#) possibly performed, and the [procedure](#) result type of the result returned by a complete and successful [execution](#) of the [procedure](#).

A procedure signature *A* matches a procedure signature *B* if and only if the procedure signature descriptor of *A* matches the procedure signature descriptor of *B*.

**** Editor's Note (number 13) ****

This topic needs to be further developed. See Possible Problem [GQL-021](#).

4.10.2.4 Procedure signature descriptor

A procedure signature is described by a *procedure signature descriptor* that comprises:

- One type signature descriptor.
- One of the types of computation described in Subclause 4.10.2.7, “Procedures classified by type of computation”, specified as follows:
 - CATALOG PROCEDURE specifies that this procedure signature is the signature of a catalog-modifying procedure.
 - PROCEDURE specifies that this procedure signature is the signature of a data-modifying procedure.
 - QUERY specifies that this procedure signature is the signature of a query.
 - FUNCTION specifies that this procedure signature is the signature of a function.

A procedure signature descriptor *A* matches a procedure signature descriptor *B* if and only if the type signature descriptor of *A* matches the type signature descriptor of *B* and the type of computation of *A* is included in the type of computation of *B*.

4.10.2.5 Type signature descriptor

A *type signature descriptor* comprises:

- One list of required mandatory procedure parameter declarations.
- One list of allowed optional procedure parameter definitions.
- One procedure result type.

A type signature descriptor *A* matches a type signature descriptor *B* if and only if the mandatory parameters of *A* match the mandatory parameters of *B*, the optional parameters of *A* match the optional parameters of *B*, and the result type of *A* is subtype of the result type of *B*.

The mandatory parameters of a type signature descriptor *A* match the mandatory parameters of a type signature descriptor *B* if and only if both *A* and *B* expect the same number of mandatory parameters *NUM* and every mandatory parameter of *A* at position *i*, $1 \leq i \leq NUM$ is a supertype of the mandatory parameter of *B* at position *i*.

The optional parameters of a type signature descriptor *A* match the optional parameters of a type signature descriptor *B* if and only if *A* expects a number of optional parameters *NUM* less than or equal to the number of optional parameters of *B* and every optional parameter of *A* at position *i*, $1 \leq i \leq NUM$ is a supertype of the optional parameter of *B* at position *i*.

**** Editor's Note (number 14) ****

Further revision required. See Possible Problem **GQL-021**.

4.10.2.6 Procedure execution

Procedures are executed by calling them in a child execution context on procedure call arguments that are passed via the working table of that execution context. The provided procedure call arguments shall fulfill the requirements of the procedure signature of the called procedure regarding the cardinality, data type, and optionality of positionally corresponding formal parameters. Every optional formal parameter of the procedure signature of the called procedure that is not included in the provided procedure call arguments is defaulted as specified by the procedure signature of the called procedure.

4.10 Procedures and commands

The parameter cardinality of binding variables is determined syntactically in the scope of the procedure containing the procedure call. A formal parameter of the procedure signature of the called procedure that is specified as requiring single parameter cardinality shall not be provided by the result of an argument expression that references iterated variables introduced prior to the procedure call.

The current working schema, working graph, and other execution context characteristics shall only be modified during the execution of the procedure as specified by its procedure descriptor.

« WG3:BER-099R1 »

The outcome of the execution *EXE* of a procedure is the execution outcome present in the specified execution context after execution has terminated. The result and status of *EXE* are the result and status, respectively, of the outcome of *EXE*.

4.10.2.7 Procedures classified by type of computation

Procedures are classified by the type of computation that their execution may perform. For this purpose, this document distinguishes between:

- A catalog-modifying procedure that is a [procedure](#) whose [execution](#) may perform [catalog-modifying side effects](#) and [data-populating side effects](#) only.
- A data-modifying procedure that is a [procedure](#) whose [execution](#) is not permitted to perform [catalog-modifying side effects](#), [session-modifying side effects](#), or [transaction-modifying side effects](#).
- A query that is a [procedure](#) whose [execution](#) may perform [data-populating side effects](#) only. A query whose execution will not access the GQL-catalog or the current GQL-session in any way is a *pure function*.

Furthermore, a [view](#) is a [query](#) that returns a [graph](#), a *simple view* is a [view](#) that expects no arguments, and a *parameterized view* is a [view](#) that expects one or more arguments.

4.10.2.8 Procedures classified by type of provisioning

Procedures are classified by how they have been provided to the GQL-server. For this purpose, this document distinguishes between:

- A GQL-procedure that is a [procedure](#) written in the GQL language that was provided directly by using the GQL language only.
- An external procedure that is a procedure provided via an implementation-defined ([IW010](#)) mechanism.

4.10.3 Commands

4.10.3.1 General description of commands

« WG3:BER-086R1 »

A [command](#) is an [operation](#) in a GQL-program that is not part of the [procedure](#) in that GQL-program.

« WG3:W21-010 P00-USA-377 »

Its execution computes an [execution outcome](#) (3.6.11) and may cause [side effects](#) (3.6.1).

Every supported command is one of the following:

- A *session command*, i.e., a [command](#) that may only perform [session-modifying side effects](#).
 - A *transaction command*, i.e., a [command](#) that may only perform [transaction-modifying side effects](#).
- « WG3:W21-010 P00-USA-378 »

NOTE 23 — Implementations may extend the set of supported kinds of commands.

4.10.3.2 Command execution

Commands are executed by invoking them in an execution context.

NOTE 24 — All arguments required by an invocation must be supplied implicitly via the current execution context.

« WG3:BER-099R1 »

The outcome of the execution *EXE* of a command is the execution outcome present in the specified execution context after execution has terminated. The result and status of *EXE* are the result and status, respectively, of the outcome of *EXE*.

« WG3:BER-086R1 »

4.10.4 GQL-procedures

4.10.4.1 Introduction to GQL-procedures

A GQL-procedure is a procedure written in the GQL language, that is part of (or available to) a GQL-program.

« WG3:BER-086R1 »

The procedure signature of a GQL-procedure is either specified explicitly by providing a <type signature> or inferred implicitly from the <procedure body> when the procedure is defined.

The procedure logic of a GQL-procedure is specified by the <procedure body> provided when the procedure is defined.

A <procedure body> comprises:

« BER-019 »

- A (possibly empty) sequence of <binding variable definition>s that are specified by a <binding variable definition block> and that define fixed variables.
- A sequence of statements that are specified by the <statement block> of the <procedure body>.

« WG3:BER-086R1 »

The procedure logic of a GQL-procedure is executed in the current execution context by first executing its variable definitions followed by executing its statements. This is specified completely by Subclause 9.3, “<procedure body>”.

This document does not specify the mechanism by which a procedure that is not a GQL-procedure is specified or provisioned to the GQL-server. Therefore, in the remainder of this document, “procedure” implies “GQL-procedure” unless otherwise specified.

4.10.4.2 Variables and parameters

Procedures interact with the following kinds of variables and parameters:

- A session parameter is a pair comprising a name and a value that is defined in a **session context**.
- A request parameter is a pair comprising a name and a value that is defined in a **GQL-request context**.

« BER-019 »

« WG3:BER-060 »

- A binding variable is a **dynamic variable** corresponding to a field name of a **working record** or a **column name** of a **working table**. Every binding variable holds a value. A variable is dynamic if its value is possibly determinable at **execution** time only. A binding variable shall be one of the following:

« BER-019 »

- A fixed variable that is a **binding variable** that is bound in the current working record.

NOTE 25 — A fixed variable is always bound to exactly one value when iterating over the current working table during procedure execution.

« BER-019 »

- An iterated variable that is a **binding variable** that is bound in the current working table.

NOTE 26 — An iterated variable may be bound to multiple values when iterating over the current working table during procedure execution.

« BER-019 Paragraph deleted »

« BER-019 List deleted »

« BER-019 List deleted »

4.10.4.3 Statements

« WG3:BER-086R1 »

A statement defines one or more operations executed as part of executing a procedure. Each operation updates the current execution context and its current execution outcome, and may cause side effects.

Many languages use a delimiter such as semicolon between the language' statements. GQL does not use any delimiter between operations within a statement; statement components are identified by leading keywords.

This document permits GQL-implementations to provide additional, implementation-defined (ID020) statements.

« WG3:BER-062 »

4.10.4.4 Statements classified by use of the current working graph

A statement is classified according to its use of the <use graph clause> as:

- 1) An *ambient statement* does not contain a <use graph clause>. An ambient statement *S* operates on the current working graph of *S*.
- 2) A *focused statement* contains one or more <use graph clause>s. A focused statement declares one or more working graphs on which it operates.

4.10.4.5 Statements classified by function

A statement is classified according to its function as:

- *catalog-modifying statement* that modifies the catalog structure to create, alter, and drop catalog objects and that cause catalog-modifying side effects to this end.
- *data-modifying statement* that performs insert, update, and delete operations on data objects and that cause data-modifying side effects to this end.
- *query statement* that in turn shall be either:
 - a *data-reading statement* that reads persistent data, or
 - a *data-transforming statement* that performs filtering, aggregating, and projecting operations.

It is possible that data-populating side effects are caused, by any statement, during the construction of new data objects.

4.10.4.6 Scope of names

« BER-019 »

The execution of procedures, statements, and commands interacts with instances such as objects and values stored at various sites. This interaction is specified indirectly by addressing those sites by the names assigned to them by their definition in the GQL-catalog, the current session context, the current request context, the current execution context, or syntactic elements such as <binding variable definition>s or the formal parameter list of a procedure. A name shall only be used in its scope, which is one or more BNF non-terminal symbols within which a name is effective.

« WG3:BER-061 »

In this document, names are always either <identifier>s, <identifier>s prefixed with some operator symbol, or but only if explicitly allowed, the zero-length character string.

**** Editor's Note (number 15) ****

The following is a high-level summary. More detailed scoping rules need to be specified. See Possible Problem [GQL-172](#)

There are five main namespaces in the GQL language:

- The *catalog namespace* of the names of every GQL-directory, GQL-schema, catalog object, or named component of a catalog object in the GQL-catalog.
- The *parameter namespace* of the names of <parameter>s resolved in order as request parameters and session parameters.

« WG3:BER-060 »

- The *local namespace* comprising <binding variable> names resolved in order as expression variables introduced by the evaluation of a <value expression>, a <graph pattern>, or a <simple graph pattern>, iterated variables introduced by iterating over working tables, fixed variables introduced by local variable definitions, formal parameters bound by a named procedure call, and catalog objects of the current working schema.
- The *label namespace* of the names corresponding to labels of catalog objects.
- The *property namespace* of the names corresponding to the properties of catalog objects and sub-objects of catalog objects that are exposed as properties.

Catalog namespace: The name of a defined GQL-directory, GQL-schema, and catalog object or named component of a catalog object is valid in the scope of every <catalog object reference>.

Parameter namespace: The name of a request parameter is valid in the scope of the containing GQL-request. The name of a session parameter is valid in the scope of a GQL-request that is executed within a session that contains such a parameter and that does not include a request parameter with the equivalent name.

Local namespace: Resolution of names in the local namespace is subject to the following rules:

- The name of a <binding variable> introduced by a <value expression> is only valid inside that expression as defined for that particular expression.
- The name of a <binding variable> introduced by a <graph pattern> or <simple graph pattern> is valid inside the statement that contains the <graph pattern> as defined for that particular statement. The exact scope of graph pattern variables is defined in [Subclause 16.5, “<graph pattern>”,](#) and in [Subclause 16.8, “<simple graph pattern>”.](#)

NOTE 27 — This includes <binding variable>s defined by <graph pattern>s and <simple graph pattern>s that are implicitly added to the current working table by the execution of a statement.

- The name of a <binding variable> introduced by a statement is valid in the scope of every sibling statement within their containing <linear query statement> or <linear data-modifying statement> until the scope is explicitly closed by a <primitive result statement>.

4.10 Procedures and commands

- The execution of a <named procedure call> explicitly interrupts the scope of every <binding variable> until after the call has finished.

« BER-019 »

- The name of a <binding variable definition> introduced in a <binding variable definition block> that is immediately contained in some <procedure body> is valid in the scope of the <statement block> immediately contained in the same <procedure body>.
- The name of a <binding variable> introduced by a formal parameter list of a procedure definition is valid in the scope of the <procedure body> that specifies the procedure logic of the procedure when the procedure is executed and interrupts the scope of names of catalog objects in the current working schema.

« BER-019 »

- The name of a catalog object in some schema is valid in the scope of every BNF non-terminal symbol that is specified to be executed with that schema as its working schema until that scope is explicitly interrupted by a formal parameter list, <binding variable definition>, statement, <value expression>, <graph pattern>, or <simple graph pattern>, as specified above.

Label namespace: Labels are valid in the scope of every BNF non-terminal symbol that is specified to be executed within a context that identifies the GQL-object whose definition implies their possible existence.

Property namespace: Properties are valid in the scope of every BNF non-terminal symbol that is specified to be executed within a context that identifies the GQL-object whose definition implies their possible existence.

4.10.5 Operations

4.10.5.1 Introduction to operations

Procedures, commands, statements, and auxiliary elements included in their specification such as <value expression>s determine the operations required for their execution and are implicitly understood to represent them where this is unambiguous in this document. Operations are either atomic or comprise suboperations. This document is only concerned with the effective execution of operations to the degree that any side effect they may cause or any result they may produce can be determined by the GQL-agent through the execution of GQL-requests or by examining the GQL-session and associated objects such as the active GQL-transaction.

NOTE 28 — This intentionally allows for a wide range of conforming GQL-implementations of this document.

Operations are classified either by their execution outcome (See Subclause 4.10.5.2, “Operations classified by execution outcome”) or by the type of side effects caused (See Subclause 4.10.5.3, “Operations classified by type of side effects”).

4.10.5.2 Operations classified by execution outcome

« WG3:BER-099R1 »

Operations such as procedures, commands, and statements that are performed as part of executing a GQL-request are classified by the type of their outcome as:

« WG3:BER-060 »

- Successful operations with a successful outcome.
- Failed operations with a failed outcome.

Execution of an operation will generally either preserve the current execution outcome that was determined just prior to the start of its execution as part of executing its containing GQL-request or procedure or will explicitly modify the current execution outcome.

4.10.5.3 Operations classified by type of side effects

« WG3:W21-010 P00-USA-395 P00-USA-396 »

Operations such as statements and commands that are performed as part of executing a GQL-request are classified by the type of side effects their execution causes as follows:

- Operations that modify the GQL-session and its context are classified as causing *session-modifying side effects*.
- Operations that manipulate GQL-transactions are classified as causing *transaction-modifying side effects*.
- Operations that modify the GQL-catalog are classified as causing *catalog-modifying side effects*.
- Operations that populate the initial content of newly created data objects are classified as causing *data-populating side effects*. Such operations may be further qualified according to whether they apply to catalog objects, session parameters, or locally-defined objects.
- Operations that populate the initial content of newly created data objects or that modify the content of existing data objects are classified as causing *data-modifying side effects*. Such operations may be further qualified according to whether they apply to catalog objects, session parameters, or locally-defined objects.
- All other operations are classified as causing no side effects.

The expression *may cause* or similar grammatical variants is used to indicate that the execution of an operation sometimes causes the specified side effects depending on a condition to be determined during the execution of the operation. The expression *always causes* or similar grammatical variants is used to indicate that the successful execution of an operation necessarily causes the specified side effects. The side effects of an operation O are all side effects SE such that O may cause SE . The side effects of a composite operation include the side effects of its suboperations, if any.

NOTE 29 — Implementations may extend GQL with additional procedures, commands, or statements that may perform additional kinds of side effects that are not considered here.

4.11 Graph pattern matching

**** Editor's Note (number 16) ****

The WG3 W16 Meeting accepted the text of WG3:W16-034R2 for application also in GQL. However, it was recognized that some enhancement would probably be necessary to cope with the enhanced facilities of GQL. See Possible Problem [GQL-239](#).

4.11.1 Summary

Graph pattern matching is the process of applying a list of special-purpose regular expressions called a <graph pattern> on a pure property graph PG in order to return a set of reduced matches. Each reduced match is a list of path bindings; each path binding is a function that maps the symbols in a word of a regular language to a path of PG . The regular language and related concepts such as path binding and reduced match are specified in Subclause 21.2, “Machinery for graph pattern matching”, and Subclause 16.5, “<graph pattern>”.

A <graph pattern> is a list of <path pattern>s. Each <path pattern> in the list is matched to PG to detect a possibly-empty set of path bindings in PG that correspond to the <path pattern>.

« WG3:BER-031 »

4.11 Graph pattern matching

The cross-product of these sets is reduced by natural joins over those global singleton element variables that are exposed by each <path pattern> and that are bound to the same graph element in PG . The remaining tuples are called reduced matches; the set of these reduced matches may be empty, but may not be infinite because of syntactic restrictions to guard against infinite cycling.

The behavior of the graph pattern matching is defined in this document.

« WG3:BER-031 »

NOTE 30 — A more detailed summary can be found in Subclause 4.11.6, “Path pattern matching”.

« WG3:BER-083R1 »

Additional qualifying parameters (predicates, a selective <path search prefix>, a <path mode>, and the <different edges match mode>) that restrict the result may be supplied.

If PG contains cycles then a match to a <path pattern> having an unbounded quantifier might return an infinite set of paths: however, this possibility is prevented by Syntax Rules that require the use of selective <path search prefix>s or restrictive <path mode>s, or <different edges match mode> (or any combination thereof) to prevent infinitely-sized result sets.

4.11.2 Paths

A *path* P is a sequence of n graph elements of a property graph PG , such that:

- n is 0 (zero) or an odd number.
- If $n = 0$ (zero) or $n = 1$ (one), then P has no edges.
- If $n \geq 1$ (one), then the graph element at each odd index is a node and the graph element at each even index is an edge that connects the pair of nodes immediately before and after the edge in the sequence.

A path contains a (possibly empty) sequence of nodes and a (possibly empty) sequence of edges. If there are two or more nodes, then the path is a sequence of graph elements that starts with a node and is followed by a sequence of ordered (edge, node) pairs.

If PG is a multigraph then an edge in the path between a pair of nodes is one of possibly several edges between those nodes in PG .

« WG3:BER-075R1 »

Every edge E in the path has an orientation. If E is undirected, then the orientation of E is *undirected*. Let $V1$ be the node that immediately precedes E in the path, and let $V2$ be the node that immediately follows E . If the source of E is $V1$ and the destination of E is $V2$, then the orientation of E is *left to right*. It is also said that the orientation of E is *directed pointing right*. If the source of E is $V2$ and the destination of E is $V1$, then the orientation of E is *right to left*. It is also said that the orientation of E is *directed pointing left*.

NOTE 31 — A directed self-edge (i.e., when E is directed and $V1$ and $V2$ are the same node), is oriented both left to right and right to left.

If no edge from PG appears more than once in a path, then the path is called a *trail*. If no node from PG appears more than once in a path, except possibly as the first and last nodes of the path, then the path is called *simple*. If no node from PG appears more than once in a path, then the path is called *acyclic*.

NOTE 32 — The term “path” is used in more than one way in mathematical graph theory and in informal technical presentations and discussions. In this document the term path always denotes what a graph-theoretic work might call a partially-oriented walk in a pure property graph. Such a graph is a mixed multigraph; edges may be directed or undirected, and there may be multiple edges between two nodes.

4.11.3 Path patterns

A <path pattern> is an expression built from the following syntactic elements, governed by the Format and Syntax Rules of Subclause 16.5, “<graph pattern>”, and other Subclauses:

« WG3:BER-031 »

- An optional <path variable declaration>, to declare a path variable to be bound to a path binding.
- An optional <path pattern prefix>.

NOTE 33 — <path pattern prefix> is described in Subclause 4.11.7, “<path mode>”, and Subclause 4.11.8, “Selective <path search prefix>”, and specified in Subclause 16.6, “<path pattern prefix>”.
- A mandatory <path pattern expression>.

A <path pattern expression> is an expression built recursively from <element pattern>s, governed by the Format and Syntax Rules of Subclause 16.7, “<path pattern expression>”, and other Subclauses.

An <element pattern> is a pattern to match a single graph element. There are two kinds of <element pattern>s:

- <node pattern>:

A <node pattern> is a pattern to match a single node. A <node pattern> comprises at a minimum a matching pair of parentheses, which may contain optional <element pattern filler>, described subsequently.

- <edge pattern>:

An <edge pattern> is a pattern to match a single edge. An <edge pattern> is either a <full edge pattern> (which optionally permits <element pattern filler>) or an <abbreviated edge pattern> (which does not support <element pattern filler>). These two major classes of <edge pattern> have seven variants each, for the seven possible non-empty combinations of the three edge orientations (the individual edge orientations being directed pointing left, undirected, or directed pointing right). Thus there are fourteen varieties of <edge pattern>.

<element pattern filler> provides three optional components of <node pattern>s and <full edge pattern>s:

« WG3:BER-031 »

- <element variable declaration>, to declare an element variable to be bound to a graph element by the <element pattern>.
- <Email from: Fred Zemke, 2022-05-22 2231 >
- <label expression>, a predicate regarding the labels of the graph element that is bound by the <element pattern>. A <label expression> is an expression formed from <label name>s and the <wildcard label> “%”, using the operation signs <vertical bar> “|” for disjunction, <ampsersand> “&” for conjunction, <exclamation mark> “!” for negation, and balanced pairs of parentheses for grouping.
- <element pattern where clause>, a <search condition> to be satisfied by the graph element that is bound by the <element pattern>.

<path pattern expression>s are regular expressions built recursively from <element pattern>s using the following operations, governed by the Format and Syntax Rules of Subclause 16.7, “<path pattern expression>”, and other Subclauses.

- concatenation, indicated syntactically by string concatenation (i.e., no operation sign).

NOTE 34 — <element pattern>s and more complex <path pattern expression>s may be concatenated in ways that appear to juxtapose either two <node pattern>s or two <edge pattern>s. These topologically inconsistent patterns are understood during pattern matching as follows:

- Two consecutive <node pattern>s must bind to the same node.
- Two consecutive <edge pattern>s conceptually have an implicit <node pattern> between them.

« Email from: Fred Zemke, 2022-05-22 2231 »

4.11 Graph pattern matching

« WG3:BER-031 »

- Grouping, using matching pairs of parentheses to form a <parenthesized path pattern expression>. A <parenthesized path pattern expression> may optionally contain the following:
 - A <subpath variable declaration> to declare a subpath variable.
 - A <search condition> to constrain matches.
- Alternation, indicated by <vertical bar> or <multiset alternation operator>.

NOTE 35 — Alternation with <vertical bar> provides set semantics using deduplication of redundant equivalent reduced matches, whereas alternation with <multiset alternation operator> provides multiset semantics, with no deduplication.
- Quantification, indicated by a postfix <graph pattern quantifier>, which may be affixed to an <edge pattern> or a <parenthesized path pattern expression>.
- <questioned path primary>, indicated by a postfix <question mark> affixed to an <edge pattern> or a <parenthesized path pattern expression>.

« WG3:BER-031 »

NOTE 36 — Unlike many regular expression languages, the <question mark> operator is not the same as {0,1}, the difference being that <questioned path primary> exposes its singleton element or subpath variables as conditional singletons, whereas {0, 1}, in common with all other quantifiers, exposes all element or subpath variables as group.

« WG3:BER-031 »

4.11.4 Graph pattern variables

A *graph pattern variable* *GPV* is a site identified by an <identifier> (the *name* of the graph pattern variable) and having a value determined by a multi-path binding *MPB* to a <graph pattern>.

There are four kinds of graph pattern variables:

- Node variables; the value of a node variable is a list of nodes.
- Edge variables; the value of an edge variable is a list of edges.
- Path variables; the value of a path variable is a path binding.
- Subpath variables.

NOTE 37 — Subpath variables are not bound to a value in this edition of this document. They serve to assure multiset semantics in <path multiset alternation>.

In a <graph table> *GT*, an <identifier> shall not identify more than one graph pattern variable; thus *GT* defines a namespace in which there is a one-to-one correspondence between the names of graph pattern variables and the graph pattern variables that they name.

**** Editor's Note (number 17) ****

The above paragraph needs to be modified for the GQL situation.

NOTE 38 — Because of this one-to-one correspondence, certain terms that are defined for graph pattern variables are also defined for their names, so that the name of the variable and the variable itself can be used interchangeably in the rules. These terms include "declare", "expose", and "degree of exposure".

Node variables and edge variables are collectively called *element variables*.

**** Editor's Note (number 18) ****

The following text from BER-031 was not included in the previous paragraph: An element variable is declared in either an <element pattern> (in which case it is a *primary element variable*) or in a <BNF name="one row per iteration"/> in which case it is an *iterator variable*).

A primary node variable *VV* and its name are declared by an <element variable declaration> simply contained in a <node pattern>.

A primary edge variable *EV* and its name are declared by an <element variable declaration> simply contained in a <full edge pattern>.

A primary variable may be declared in more than one <element pattern>. A multiply declared primary variable *MDPV* expresses a natural equijoin in two circumstances:

- If *MDPV* is declared in both operands of a <path concatenation>.
- If *MPDV* is declared in two or more <path pattern>s of a <graph pattern>.

NOTE 39 — Declaring an element variable in two operands of a <path pattern union> or <path multiset alternation> does not express a natural equijoin.

**** Editor's Note (number 19) ****

The following paragraphs from BER-031 was not included:

<BNF name="one row per vertex"/> declares a single iterator vertex variable and its name.

<one row per step> declares an iterator vertex variable, an iterator edge variable, and another iterator vertex variable, as well as their names.

Iterator element variables may not be multiply declared.

A path variable *PV* and its name are declared by a <path variable declaration> simply contained in a <path pattern>. A path variable may only be declared once.

A subpath variable *SV* and its name are declared by a <subpath variable declaration> simply contained in a <parenthesized path pattern expression>. More than one operand of a <path pattern union> may declare *SV*; otherwise *SV* may not be multiply declared.

4.11.5 References to graph pattern variables

Graph pattern variables are visible within the <graph table> *GT* in which they are declared. They may be referenced in scalar expressions and <search condition>s within *GT*.

**** Editor's Note (number 20) ****

The above paragraph needs to be modified for the GQL situation.

**** Editor's Note (number 21) ****

The following paragraph from BER-031 was not included:

Iterator variables and path variables may only be referenced in <graph table column definition>s.

Primary variables may be referenced in the following BNF non-terminals:

- <element pattern where clause>
- <parenthesized path pattern where clause>
- <graph pattern where clause>

**** Editor's Note (number 22) ****

The following item from BER-031 was not included: <graph table column definition>

If a primary element variable *PEV* is declared in a <quantified path primary> *QPP*, then it may bind to more than one graph element. References to *PEV* are interpreted contextually: if the reference occurs

4.11 Graph pattern matching

outside *QPP*, then the reference is to the complete list of graph elements that are bound to *PEV*. In this circumstance *PEV* is said to have *group degree of reference*. If the reference does not cross a quantifier, then the reference has *singleton degree of reference* and references at most one graph element, even if the multi-path binds *PEV* multiple times.

NOTE 40 — For example

```
(X) -[E WHERE E.P > 1]->{1,10} (Y) WHERE SUM(E.P) < 100
```

This example references primary edge variable *E* twice: once in the <edge pattern> and once outside the <edge pattern>. Within the <edge pattern>, *E* has singleton degree of reference and the <property reference> *E.P* references a property a single edge. On the other hand, the reference within the *SUM* aggregate has group degree of reference (because of the quantifier {1,10}) and references the list of edges that are bound to *E*.

A reference *R* to a graph pattern variable *GPV* is termed *local* in these circumstances:

- If *GPV* is declared in an <element pattern> *EP* and *R* is contained in the <element pattern where clause> of *EP*.
- If *GPV* is declared in a <parenthesized path pattern expression> *PPPE* and *R* is contained in the <parenthesized path pattern where clause> of *PPPE*.
- If *R* is in a <graph pattern where clause>.

**** Editor's Note (number 23) ****

The following text from BER-031 was not included: or a <graph table column definition>

R has a degree of reference, determined by the Syntax Rules of Subclause 17.11, “<element reference>”. The degree of reference of *R* is one of the following: unconditional singleton, conditional singleton, or effectively bounded group.

**** Editor's Note (number 24) ****

The reference to the Subclause for "path reference" from BER-031 was not included in the paragrpah above.

NOTE 41 — Subpath variables cannot be referenced by SQL language defined by this document, though they may be referencable in a future version. At present their only use is to distinguish operands of a <path multiset alternation>.

**** Editor's Note (number 25) ****

It is a Language Opportunity to support references to subpath variables, for example, in <graphical path length function>, or a TOTAL_COST function once CHEAPEST is defined. See Language Opportunity [GQL-279](#).

NOTE 42 — In general, an element variable that is declared within a <quantified path primary> is bound by a multi-path binding to a list of graph elements. The reference *R*, on the other hand, may reference a proper subset of this list, based on the syntactic context in which *R* is placed. The degree of reference expresses the cardinality of the list that *R* references, as follows: an unconditional singleton references a list of length 1 (one), a conditional singleton references a list of length 0 (zero) or 1 (one), an effectively bounded group references a finite list. Syntax rules prohibit the possibility of referencing an infinite list.

A reference to a path variable always has unconditional singleton degree of reference.

References to graph pattern variables are subject to the following constraints:

- An operand *OP* of <path pattern union> or <path multiset alternation> *U* may only reference element variables declared in *OP*, or outside of *U*.
- A non-local reference shall have singleton degree of reference.
- A group reference shall be contained in an aggregated argument of a <set function specification>. The group references in an aggregated argument of a <set function specification> shall reference the same graph pattern variable. All other references to graph pattern variables in a <set function specification> shall have singleton degree of reference.

**** Editor's Note (number 26) ****

The above item needs to be modified/deleted for GQL.

- A selective <path pattern> *SPP* shall not reference a graph pattern variable that is not declared in *SPP*.

**** Editor's Note (number 27) ****

The following item from BER-031 was not included: Iterator variables shall only be referenced in <graph table column definition>s.

**** Editor's Note (number 28) ****

The following paragraph from BER-031 was not included):

If *GT* has a <graph table export clause> other than EXPORT NO SINGLETONS, then certain singleton <graph pattern variable>s (the exported <graph pattern variable>s) are visible in the <query specification> whose <from clause> simply contains *GT*. A syntactic transformation defines an equivalent <query specification> in which the <graph table> specifies EXPORT NO SINGLETONS, and all references to exported <graph pattern variable>s have been placed in <graph table column definition>s.

4.11.6 Path pattern matching

Path pattern matching is performed by Subclause 16.5, “<graph pattern>”, which in turn may invoke Subclause 21.3, “Evaluation of a <path pattern expression>”, and Subclause 21.4, “Evaluation of a selective <path pattern>”, as well as other Subclauses incidentally invoked for expression evaluation.

« WG3:BER-031 deleted one Editor's Note »

« WG3:BER-031 »

In more detail, each <path pattern> of a <graph pattern> is evaluated independently of each other, resulting in a set of path bindings. A path binding is a list of elementary bindings; each elementary binding is an ordered pair (*LET*, *GE*), where *LET* is a member of the alphabet, which comprises the element variable names, plus additional special symbols for the anonymous node symbol, the anonymous edge symbol, bracket symbols to indicate the beginning and ending of bindings to <parenthesized path pattern expression>s, and subpath symbols to mark the beginning and ending of subpaths.

NOTE 43 — In the formal semantics, <label expression>s are evaluated at this stage, but <search condition>s are not; hence there may be path bindings that will be subsequently rejected because they fail a <search condition>. Implementations are of course free to “push down” predicate evaluation as long as the ultimate results are the same as prescribed by the formal semantics.

Projecting the elementary bindings of a path binding to the first component yields a word of the regular language of the <path pattern>. Projecting to the second component yields an annotated path, which is a path interspersed with mark-up to indicate the beginning and ending of <parenthesized path pattern expression>s and the beginning and ending of subpaths.

« WG3:BER-083R1 »

If a <path pattern> has an unbounded quantifier that is not in the scope of a restrictive <path mode> or <different edges match mode>, there may be infinitely many path bindings. Such a <path pattern> must have a selective <path search prefix> *SPSP*. Subclause 21.4, “Evaluation of a selective <path pattern>”, is invoked to reduce this potentially infinite set of path bindings to a finite set. All <search condition>s contained in the selective <path pattern> are evaluated to reduce the set of path bindings prior to making the final selection according to *SPSP*.

NOTE 44 — An implementation cannot generate an infinite set and then apply <search condition>s; instead it must enumerate the search space in a fashion enabling it to arrive at the same result as specified by the formal semantics. The formal semantics do not specify the algorithm for this enumeration, only the result.

« WG3:BER-083R1 »

4.11 Graph pattern matching

At this point, there is a finite set of path bindings for each *<path pattern>* in a *<graph pattern>*. The cross product of these sets is formed; a member of the cross product is called a multi-path binding. A multi-path binding does not violate any restrictive *<path mode>* or *<different edges match mode>* that may be in force. The set of multi-path bindings is reduced by enforcing natural equijoins on the unconditional singleton variables exposed by the *<path pattern>*s.

Next the *<search condition>*s in the *<graph pattern>* are evaluated, potentially further reducing the set of multi-path bindings. Those that remain satisfy all the selective *<path search prefix>*es and all the *<search condition>*s of the *<graph pattern>*.

Next the function *REDUCE* (defined in Subclause 21.2, “Machinery for graph pattern matching”) is applied to the remaining multi-path bindings. *REDUCE* removes the elementary bindings of the bracket symbols, and collapses adjacent anonymous node symbol bindings into a single elementary binding. The results, now called reduced matches, are deduplicated according to set semantics.

NOTE 45 — Duplicates can arise if there is a *<path pattern union>* in the *<graph pattern>*.

4.11.7 *<path mode>*

A *<path mode>* may be specified for any *<parenthesized path pattern expression>* or any *<path pattern>* from the following choices:

- WALK, the default *<path mode>*, is the absence of any filtering implied by the other *<path mode>*s.
- TRAIL, where path bindings with repeated edges are not returned.
- ACYCLIC, where path bindings with repeated nodes are not returned.
- SIMPLE, where path bindings with repeated nodes are not returned unless these repeated nodes are the first and the last in the path.

Using trail, acyclic, or simple matching path modes for all unbounded quantifiers guarantees that the result set of a graph pattern matching will be finite.

4.11.8 Selective *<path search prefix>*

The set of path bindings resulting from a graph pattern match can be further restricted by a selective *<path search prefix>* *SPSP*. *SPSP* is defined by partitioning the potentially infinite set of path bindings by the endpoints, which are the first and last nodes bound by the path bindings.

NOTE 46 — This partitioning is crucial to the definition of *SPSP*. *SPSP* makes a selection of some number of path bindings from each partition. For example, a path binding is “shortest” if its length is minimal within its partition. A “shortest” path binding in one partition may be longer than a “shortest” path binding in another partition.

SPSP can constrain the result set in the following ways:

- *<any path search>*: to non-deterministically pick some number of path bindings from each partition; the number is specified by an *<unsigned integer specification>*.
- *<all shortest path search>*: to pick all the shortest path bindings in each partition.
- *<counted shortest path search>*: to non-deterministically pick some number of shortest path bindings from each partition; the number is specified by an *<unsigned integer specification>*.
- *<counted shortest group search>*: to group each partition into groups of path bindings having the same length, order the groups by path length, and pick all path bindings in some number of groups from the front of each partition; the number is specified by an *<unsigned integer specification>*.

The specification of *SPSP* guarantees that the result set of a graph pattern matching will be finite.

« WG3:BER-032R3 »

4.11.9 <match mode>

A <graph pattern> GP may optionally specify a <match mode> that applies to all <path pattern>s simply contained in GP .

There are two <match mode>s:

- DIFFERENT EDGES: A matched edge may not bind to more than one edge variable. There are no restrictions on matched nodes.
- REPEATABLE ELEMENTS: There are no restrictions on matched edges or matched nodes.

If a <match mode> is not specified, then an implementation-defined (ID086) <match mode> is implicit.

4.12 Data types

** Editor's Note (number 29) **

WG3:W05-020 suggests that, whilst the introductory text is based on that in SQL/Foundation, the Subclause as a whole has not yet reached consensus agreement. See Possible Problem [GQL-098](#).

** Editor's Note (number 30) **

Inclusion of null in every data type requires further discussion. See Possible Problem [GQL-018](#).

4.12.1 General introduction to data types

« WG3:BER-040R3 »

A *data type* is a set of elements with shared characteristics representing data. An *object type* is a data type comprising *objects*, a *value type* is a data type comprising *values*, a *reference value type* is a value type comprising *reference values*, and a *property value type* is a value type supported as the type of *property values*. A data type characterizes the sites that may be occupied by instances of its elements. Every instance at a site belongs to one or more data types. The physical representation of an instance of a data type is implementation-dependent (UV005).

A data type A may be equivalent to, disjoint from, overlap with, or be included in some other data type B depending on the sharing of elements between A and B . In this document, any two data types with the exact same elements are considered equivalent. A data type containing all elements of some data type X is a supertype of X . A data type comprising elements contained in some data type X is a subtype of X . In the context of data types, the verb “to be” (including all its grammatical variants, such as “is”) is defined as follows: A data type T is said to be data type U if T is a subtype of U . All data types form a partial type hierarchy that is implicitly defined by the subtyping relation between them.

A *most specific type* of some element X is a smallest supertype of all data types supported by the GQL-implementation that include X . In this document, this data type is always either uniquely defined or determined to be the normal form of all equivalent data types containing a material X . The *strict elements* of a data type are the elements for which that data type is the most specific type.

A *base type* is a set of data types with shared characteristics (i.e., “type of types” such as the base type of all character string types). Every data type is of exactly one single associated base type that classifies the data type itself. Base types implicitly determine the universe of elements from which each of their data types are drawn (e.g., the integers). Every base type has a name. The name of a base type is a sequence of reserved words that is specified by this document that specifies the base type in specific contexts (such as base type names in data type descriptors).

« WG3:BER-040R3 »

The name of a base type is always formed from a *base type name prefix* that is followed by either the reserved word DATA or the reserved word REFERENCE. Elements of a data type of some base type A

may still be assignable to a site whose declared type is another data type of some other base type *B* that is different from *A* through the application of Rules for implicit type conversion.

This document defines the following kinds of data types:

- A *predefined* data type (such as a *character string type*) is a data type specified by this document that is **atomic** and provided by the GQL-implementation. A data type is predefined even though the user is required (or allowed) to provide certain parameters when specifying it (for example the precision of a number). Predefined data types are sometimes called “built-in data types”, though not in this document.
- A *constructed* data type such as a *record type* is a data type comprising **composite elements**.
- A *user-defined* data type is a data type that is **constructed** and explicitly defined by the user. User-defined data types can be defined by a standard, by a GQL-implementation, or by an application.

« WG3:BER-074 »

An *atomic* data type is a data type comprising only values that are not composed of values of other data types. The existence of an operation (such as SUBSTRING) that is capable of selecting part of an element of some data type *T* (such as a character string or a datetime value) does not imply that *T* is not an atomic data type.

A *material* data type is a data type excluding the **null value**. Two material data types of different base types are always disjoint. A *nullable* data type is a data type including the **null value**. In this document, the nullability characteristic of a site is recorded in the data type descriptor of the declared type of the site. The null values of all nullable data types are identical.

**** Editor's Note (number 31) ****

Equality rules with different treatment of *NULL* to be defined. See Possible Problem [GQL-173](#).

**** Editor's Note (number 32) ****

Comparison rules with different treatment of *NULL* to be defined. See Possible Problem [GQL-174](#).

4.12.2 Naming of predefined value types and associated base types

Every predefined value type is specified by any of its *declared names*, including its *preferred name*, which is determined subject to implementation-defined provisions. A declared name is a non-empty sequence of reserved words specified by this document. The names of all base types of all predefined value types are specified by one of the following sequences of <key word>s:

- BOOLEAN DATA
- STRING DATA
- BINARY DATA
- INTEGER DATA
- DECIMAL DATA
- FLOAT DATA

For reference purposes:

- The value types specified by BOOL and BOOLEAN are referred to as *Boolean types* and the values of Boolean types are referred to as *truth values*, or, alternatively, as *Booleans*. The base type of all Boolean types is BOOLEAN DATA.

- The value types specified by STRING and VARCHAR are referred to as *character string types* and the values of character string types are referred to as *character strings*. The base type of all character string types is STRING DATA.
- The value types specified by BYTES, BINARY, and VARBINARY are referred to as *byte string types* and the values of byte string types are referred to as *byte strings*. The base type of all byte string types is BINARY DATA.
- Exact numeric types and approximate numeric types are collectively referred to as *numeric types*. Values of numeric types are referred to as *numbers*.
- The signed exact numeric types and the unsigned exact numeric types are collectively referred to as *exact numeric types*. Values of exact numeric types are referred to as *exact numbers*.

« WG3:BER-067R1 »

- The value types specified by DECIMAL, DEC, SMALLINT, SMALL INTEGER, SIGNED SMALL INTEGER, INT, INTEGER, SIGNED INTEGER, INT16, INTEGER16, SIGNED INTEGER16, INT32, INTEGER32, SIGNED INTEGER32, INT64, INTEGER64, SIGNED INTEGER64, INT128, INTEGER128, SIGNED INTEGER128, INT256, INTEGER256, SIGNED INTEGER256, BIGINT, BIG INTEGER, and SIGNED BIG INTEGER are collectively referred to as *signed exact numeric types*. Values of signed exact numeric types are referred to as *signed exact numbers*.
- The value types specified by USMALLINT, UNSIGNED SMALL INTEGER, UINT, UNSIGNED INTEGER, UINT16, UNSIGNED INTEGER16, UINT32, UNSIGNED INTEGER32, UINT64, UNSIGNED INTEGER64, UINT128, UNSIGNED INTEGER128, UINT256, UNSIGNED INTEGER256, UBIGINT, and UNSIGNED BIG INTEGER are collectively referred to as *unsigned exact numeric types*. Values of unsigned exact numeric types are referred to as *unsigned exact numbers*.
- Exact numeric types with binary precision in bits and a scale of 0 (zero) are collectively referred to as (signed or unsigned) *integer types*. Values of (signed or unsigned) integer types are referred to as (signed or unsigned) *integer numbers*, or, alternatively as (signed or unsigned) *integers*. The base type of all (signed or unsigned) integer types is INTEGER DATA.
- Exact numeric types with decimal precision and scale in digits are referred to as *decimal types*. Values of decimal types are collectively referred to as *decimal numbers*. The base type of all decimal types is DECIMAL DATA.
- The value types specified by FLOAT, FLOAT16, FLOAT32, FLOAT64, FLOAT128, FLOAT256, REAL, DOUBLE, and DOUBLE PRECISION are collectively referred to as *approximate numeric types* and the values of approximate numeric types are known as *approximate numbers*, or, alternatively, as *floating point numbers*. The base type of all approximate numeric types is FLOAT DATA.

4.12.3 Data type descriptors

Each data type has an associated data type descriptor; the content of a data type descriptor are determined by the specific data type that it describes. A data type descriptor includes an identification of the data type and all information needed to characterize an element of that data type.

Subclause 20.2, “<value type>”, describes the semantic properties of each value type.

4.12.4 Data type terminology

Two data types, T_1 and T_2 , are said to be compatible if T_1 is assignable to T_2 , T_2 is assignable to T_1 , and their descriptors include the same data type name. If they are record types, it shall further be the case that the declared types of their corresponding fields are pairwise compatible.

« WG3:BER-094R1 »

If they are list types, it shall further be the case that their element types are compatible.

4.12.5 Properties of distinct

Two comparable values are distinct if they are capable of being distinguished within a given context.

Two null values are not distinct.

A null value and a material value are distinct.

The application of the General Rules of Subclause 21.7, “Determination of distinct values”, determines whether two comparable values are distinct or not.

It is undefined whether two values that are not comparable are distinct or not.

4.13 Type hierarchy outline

**** Editor's Note (number 33) ****

WG3:W05-020 suggests that the text in this Subclause has not been considered by WG3 and that consensus agreement is required. See Possible Problem [GQL-061](#).

4.13.1 Introduction to type hierarchy outline

The *type hierarchy* forms an algebraic lattice of data types that puts the “any type” at the top and the “nothing type” at the bottom.

4.13.2 The any type

**** Editor's Note (number 34) ****

This Subclause requires further discussion. See Possible Problem [GQL-061](#).

The *any type* is the largest supported data type and therefore is placed at the root of the type hierarchy.

Type characteristics: The any type is a taxonomic data type.

Subtyping: Every data type is a subtype of the any type.

4.13.3 The any object type

**** Editor's Note (number 35) ****

This Subclause requires further discussion. See Possible Problem [GQL-061](#).

The *any object type* is the supertype of all objects.

Type characteristics: The any object type is a taxonomic data type.

Subtyping: Every object type is a subtype of the any object type.

4.13.4 The any value type

**** Editor's Note (number 36) ****

This Subclause requires further discussion. See Possible Problem [GQL-061](#).

The *any value type* is the supertype of all values.

Type characteristics: The any value type is a taxonomic data type.

Subtyping: Every value type is a subtype of the any value type.

4.13.5 Union types

** Editor's Note (number 37) **

If GQL decides to adopt dynamic type checking as its typing discipline, then union types should be removed. See Possible Problem [GQL-007](#).

A *union type* between two data types T and U is the union of every element of T and every element of U .

Type characteristics: A union type between a data type T and a data type U is the same data type as a union type between U and T . A union type between a data type T and a data type U is the same data type as U if and only if T is a subtype of U .

Subtyping: Any data type T is a subtype of a union type between T and some other data type.

NOTE 47 — Every data type is a subtype of every data type between itself and some other data type.

4.13.6 The any property value type

** Editor's Note (number 38) **

This Subclause requires further discussion. See Possible Problem [GQL-061](#).

The *any property value type* is the supertype of every property value of every graph element of GQL-data.

Type characteristics: The any property value type is a taxonomic data type.

Subtyping: The any property value type is the supertype of all predefined value types and all regular list types of lists of elements that are all subtypes of the property value type.

NOTE 48 — Implementations may extend the definition of the any property value type to encompass additional value types.

4.13.7 The nothing type

** Editor's Note (number 39) **

This Subclause requires further discussion. See Possible Problem [GQL-061](#).

The *nothing type* is the only supported empty data type. It is the smallest data type and therefore is placed at the bottom of the type hierarchy. The nothing type has no elements.

Type characteristics: The nothing type is an abstract taxonomic data type.

Subtyping: The nothing type is a subtype of every other data type.

4.14 Graph types

** Editor's Note (number 40) **

WG3:W05-020 suggests that, although the contents of this Subclause has been discussed, consensus has not yet been reached on all aspects. Some of the material is based on [WG3:SXM-030r3] and [WG3:MMX-028r2]. This is indicated for each subsection. It has been touched on in [WG3:W04-013] and [WG3:MMX-079r1], and is part of the road map in [WG3:W02-014]. Consensus agreement is required. See Possible Problem [GQL-062](#)

4.14.1 Introduction to graph types

** Editor's Note (number 41) **

This Subclause was accepted and adapted from [WG3:SXM-030r3] except for the material on type characteristics and subtyping, but should be reviewed. See Possible Problem [GQL-062](#).

A *graph type* describes the attributes of and the nodes and edges that may occur in a conforming graph. A graph type comprises:

- A graph type label set that comprises zero or more labels. A label has a name, which is an identifier that is unique within the graph type.

The maximum cardinality of graph type label sets is implementation-defined ([IL007](#)).

Without Feature GA00, “Graph label set support”, the graph type label set of a conforming graph type shall be empty.

- A graph type property type set that comprises a set of zero or more property types. Each property type comprises:

- Its property name, which is an identifier that is unique within the graph type.

NOTE 49 — The names of graph type labels and of graph type property types are in separate namespaces.

- Its property value type, which is a subtype of the any property value type.

The maximum cardinality of graph type property type sets is implementation-defined ([IL008](#)).

Without Feature GA01, “Graph property set support”, the graph type property type set of a conforming graph type shall be empty.

- A set of node types.

- A set of edge types.

- A node type name dictionary that maps node type names, which are identifiers, to node types such that each node type name is mapped to a single node type. These node types shall be contained in the same graph type.

Without Feature GA08, “Named node types in graph types”, a conforming graph type shall contain an empty node type name dictionary.

- An edge type name dictionary that maps edge type names, which are identifiers, to edge types such that each edge type name is mapped to a single edge type. These edge types shall be contained in the same graph type.

Without Feature GA09, “Named edge types in graph types”, a conforming graph type shall contain an empty edge type name dictionary.

NOTE 50 — A graph type may be associated with a catalog object that has a name, but graph types can exist that have no name, or at least have no name that is required to be visible to a user of a GQL-implementation. A graph projected by a query has a type, but it is not cataloged or named.

The element types of a graph type may be references to element types defined in another graph type.

« WG3:BER-082R1 deleted one Note »

**** Editor's Note (number 42) ****

Details of element type aliasing need to be specified. See Possible Problem [GQL-098](#).

Type characteristics: Every graph type is a constructed type.

Subtyping: Every graph type is a subtype of the any object type.

Any graph type T is a subtype of any graph type U if and only if:

- The graph type label set of T contains the graph type label set of U .
- The graph property type set of T contains the graph property type set of U .

- Every node type in the node type set of T is a subtype of some node type from the node type set of U .
- Every edge type in the edge type set of T is a subtype of some edge type from the edge type set of U .

4.14.2 Graph type descriptors

** Editor's Note (number 43) **

The text in this section is based on [WG3:MMX-028r2]. Consensus agreement is needed. See Possible Problem [GQL-062](#).

A graph type is described by a graph type descriptor that includes:

- The name of the graph type (also known as the graph type name) that is the name of the catalog object.
- A set of zero or more labels (also known as a graph type label set). A label has a name that is an identifier that is unique within the graph type label set.
- A set of zero or more property type descriptors (also known as a graph type property type set).
- A set of node type descriptors (also known as a node type set).
- A set of edge type descriptors (also known as an edge type set).
- A node type name dictionary that maps node type names, which are identifiers, to node types such that each node type name is mapped to a single node type. These node types shall be contained in the graph type node type set of the same graph type descriptor.
- An edge type name dictionary that maps edge type names, which are identifiers, to edge types such that each edge type name is mapped to a single edge type. These edge types shall be contained in the graph type edge type set of the same graph type descriptor.

Two graph type descriptors describe equal graph types if they contain:

- Equal graph type label sets.
- Equal graph type property type sets.
- Equal node type sets.
- Equal edge type sets.
- Equal node type name dictionaries.
- Equal edge type name dictionaries.

Two node and edge type name dictionaries are equal if

- They map an equal set of node and edge type name and
- Map each of these node and edge type names to equal node and edge types, respectively.

A *named graph type descriptor* is both a catalog object descriptor and a graph type descriptor and comprises every component of both kinds of descriptors.

4.14.3 Graph element types

** Editor's Note (number 44) **

Base element and refined types need to be further aligned. In particular, base element types need to be considered by Subclause 4.14, "Graph types". This note applies to all of the Subclauses of Subclause 4.14.3, "Graph element types". See Possible Problem [GQL-098](#).

4.14.3.1 Graph element base type

A graph element is either a node or an edge. The *graph element base type* is the data type of all graph elements

Subtyping: The graph element base type is the supertype of every graph element type.

4.14.3.2 Node base type

A node is the fundamental unit from which a graph is formed. The *node base type* is the data type of all nodes.

Subtyping: The node base type is a subtype of the graph element base type. The base node type is the supertype of every node type.

Identity: Two nodes are identical if they represent the same original node from the same base graph.

4.14.3.3 Node types

A *node type* is the data type of nodes that have specific labels and that have specific properties and whose properties have a specific property value type.

Each node type comprises:

- A node type label set that comprises a set of zero or more labels. A label has a name that is an identifier that is unique within the node type.

The maximum cardinality of node type label sets is implementation-defined ([IL009](#)).

Without Feature GA02, "Empty node label set support", a node type in a conforming graph type shall contain a non-empty node type label set.

Without Feature GA03, "Singleton node label set support", a node type in a conforming graph type shall not contain a singleton node type label set.

Without Feature GA04, "Unbounded node label set support", a node type in a conforming graph type shall not contain a node type label set that comprises one or more labels.

- A node type property type set that comprises a set of zero or more property types. Each property type comprises:

- Its property name that is an identifier that is unique within the node type.

NOTE 51 — The names of node type labels and of node type property types are in separate namespaces.

- Its property value type that is a subtype of the any property value type.

The maximum cardinality of node type property type sets is implementation-defined ([IL010](#)).

« WG3:BER-040R3 »

A GQL-implementation is permitted to regard certain <node type definition>s as equivalent, if they have the same label set and property type set, as permitted by the Syntax Rules of Subclause 13.11, "<label set definition>", and Subclause 13.12, "<property type set definition>". When two or more <node type definition>s are equivalent, the GQL-implementation chooses one of these equivalent <node type definition>s as the normal form representing that equivalence class of <node type definition>s. The normal form determines the preferred syntax of the node type in data type descriptors.

Strict elements: Every node that has the exact set of labels and properties required by the node type and whose property values strictly conform to the corresponding property value types required by the node type is a strict of the node type.

Subtyping: A node type is a supertype of another node type if the other node type requires the same restrictions on labels, properties, and property value types.

4.14.3.4 Node type descriptor

A node type is described by the node type descriptor. The node type descriptor includes:

- The node type name, if specified.

** Editor's Note (number 45) **

The consequences of including the name need further consideration. See Possible Problem [GQL-038](#).

- A set of zero or more labels (also known as a node type label set). A label has a name that is an identifier that is unique within the node type label set.

- A set of zero or more property type descriptors (also known as a node type property type set).

Two node type descriptors describe equal node types if they contain equal node type label sets and equal node type property type.

4.14.3.5 Edge base type

An edge has zero or more labels and zero or more properties and represents a connection from a source node to a destination node in the same graph (if directed) or a connection between two endpoints (if undirected). The *edge base type* is the data type of all edges.

Subtyping: The edge base type is a subtype of the graph element base type. The edge base type is the supertype of every edge type.

An edge type that is introduced by an edge type definition of a graph type restricts its elements in terms of their required directionality, required labels and properties, and the required property value types of those properties, as well as the required node types of their endpoints.

Identity: Two edges are identical if they represent the same original edge from the same base graph.

4.14.3.6 Edge types

An *edge type* is the data type of edges that have specific labels and that have specific properties and whose properties have a specific property value type and whose endpoints conform to specific node types.

Each edge type comprises:

- An edge type label set that comprises a set of zero or more labels. A label has a name that is an identifier that is unique within the edge type.

The maximum cardinality of edge type label sets is implementation-defined ([IL011](#)).

Without Feature GA05, “Empty edge label set support”, an edge type in a conforming graph type shall contain a non-empty edge type label set.

Without Feature GA06, “Singleton edge label set support”, an edge type in a conforming graph type shall not contain a singleton edge type label set.

Without Feature GA07, “Unbounded edge label set support”, an edge type in a conforming graph type shall not contain an edge type label set that comprises one or more labels.

4.14 Graph types

- An edge type property type set that comprises a set of zero or more property types. Each property type comprises:
 - Its property name that is an identifier that is unique within the edge type.

NOTE 52 — The names of edge type labels and of edge type property types are in separate namespaces.

 - Its property value type that is a subtype of the any property value type.

The maximum cardinality of edge type property type sets is implementation-defined ([IL012](#)).
 - Two (possibly identical) endpoint node types that are node types contained in the same graph type.
 - An indication whether the edge type is a directed edge type or an undirected edge type.
- Additionally, a directed edge type identifies one of its endpoint node types as its source, and the other as its destination. The direction of a directed edge type is from its source to its destination.
- Without Feature GA10, “Undirected edge patterns”, a conforming graph type shall not contain undirected edge types.

« WG3:BER-040R3 »

A GQL-implementation is permitted to regard certain <edge type definition>s as equivalent, if they have the same endpoints, label set and property type set, as permitted by the Syntax Rules of Subclause 13.10, “<edge type definition>”, Subclause 13.11, “<label set definition>”, and Subclause 13.12, “<property type set definition>”. When two or more <node type definition>s are equivalent, the GQL-implementation chooses one of these equivalent <node type definition>s as the normal form representing that equivalence class of <node type definition>s. The normal form determines the preferred syntax of the node type in data type descriptors.

Strict elements: Every edge that has the exact set of labels and properties required by the edge type and whose property values strictly conform to the corresponding property value types required by the edge type and whose source nodes strictly conform to the node types required for source nodes of the edge type and whose destination nodes strictly conform to the node types required for endpoints of the edge type (if directed) or whose endpoints strictly conform to the node types required for endpoints of the edge type (if undirected) is a strict element of the edge type.

Subtyping: An edge type is a supertype of another edge type if the other edge type requires the same restrictions on labels, properties, property value types, directionality, and the node types of endpoints.

4.14.3.7 Edge type descriptor

An edge type is described by the edge type descriptor. The edge type descriptor includes:

- The edge type name, if specified.

**** Editor's Note (number 46) ****

The consequences of including the name need further consideration. See Possible Problem [GQL-038](#).

- A set of zero or more labels (also known as an edge type label set). A label has a name that is an identifier that is unique within the edge type label set.
- A set of zero or more property type descriptors (also known as an edge type property type set).
- An indication whether the edge type is directed or undirected.
- Case:
 - If the edge type is directed, then:

- A source node type descriptor.
- A destination node type descriptor.
- If the edge type is undirected, then a set of two endpoint node types descriptors.

Two edge type descriptors describe equal edge types if they contain

- Equal edge type label sets,
- Equal node type property type,
- Equal indication whether they are directed or undirected, and
- Case:
 - If the edge types are directed, then:
 - Equal source node types and
 - Equal destination node types.
 - If the edge types are undirected, then equal sets of endpoint node types.

4.14.3.8 Property types

4.14.3.8.1 Introduction to property types

A property type is a pair comprising:

- The property name that is an identifier.
- A declared type that can be any property value type.

Two property types are equal if they have equal property names and equal declared types.

4.14.3.8.2 Property type descriptor

Each property type descriptor comprises:

- The property name that is an identifier.
- A declared type that can be any property value type.

4.15 Binding table types

** Editor's Note (number 47) **

WG3:W05-020 suggests that, despite being touched on in [WG3:W04-013], further discussion needed to establish consensus agreement. See Possible Problem [GQL-063](#).

« BER-019 »

A binding table type is the object type of a binding table. Every binding table type is described by a binding table data type descriptor. A binding table data type descriptor comprises:

- The name of the base type of all binding table types (BINDING TABLE DATA).
[« WG3:BER-040R3 »](#)
- A closed material record type.
- An indication of whether the type contains the null value.

For every binding table type *BTT* with a record type *RT*, a binding table *BT* is a GQL-object of *BTT*, if and only if every record *R* contained in *BT* is of *RT*.

In this document, a column is a field type of the record type of a binding table type, a column name is a field name of a column, and a column type is a field value type of a column. Hence, a column name identifies the fields of the same name of the records of a binding table.

« WG3:BER-040R3 »

A GQL-implementation is permitted to regard certain <binding table type>s as equivalent, if they have the same record type and indication regarding the inclusion of the null value, as permitted by the Syntax Rules of Subclause 11.4, “<binding table type>”. When two or more <record type>s are equivalent, the GQL-implementation chooses one of these equivalent <record type>s as the normal form representing that equivalence class of <binding table type>s.

Further, a *unit binding table type* is the binding table type whose record type is a unit record type. The binding table type of the unit binding table is a unit binding table type.

The binding table type of an empty binding table is also a unit binding table type unless specified otherwise.

« WG3:BER-040R3 deleted four paragraphs and a Note »

4.16 Value types

4.16.1 Constructed types

« BER-019 »

**** Editor's Note (number 48) ****

WG3:W05-020 suggests that, despite being discussed in [WG3:W03-034] with the exception of Subclause 4.16.1.3, “Path type” which is included in the road map in [WG3:W02-014] and touched in [WG3:W04-013], further discussion is needed to establish consensus agreement. See Possible Problem **GQL-064**.

4.16.1.1 Introduction to constructed types

« WG3:BER-094R1 »

The GQL language supports constructed types that are either list types or map types.

« WG3:BER-094R1 deleted one paragraph »

A collection may be empty.

The GQL language supports maps that may be regular maps or records, and that may be empty.

« WG3:BER-094R1 deleted two Subclauses »

4.16.1.2 List types

4.16.1.2.1 List types

A value of *list type* with element type *ET* is an ordered collection of elements of *ET* that is called a *list* or an *array*. In a list *L*, each element is associated with exactly one ordinal position in *L*. If *n* is the cardinality of *L*, then the ordinal position *p* of an element is an integer in the range 1 (one) $\leq p \leq n$.

Type characteristics: A list type with element type *T* is an abstract value type if and only if *T* is an abstract value type.

Strict elements: A list *L* is a strict element of a list type with element type *T* if and only if *T* is the join of the most specific types of all elements of *L*.

Subtyping: Every list type with element type T is a subtype of a collection type with element type U if and only if T is a subtype of U . Every list type with element type T is a subtype of a list type with element type U if and only if T is a subtype of U . Every distinct list type with element type T is a subtype of a list type with element type U if and only if T is a subtype of U . Every distinct list type with element type T is a subtype of a distinct list type with element type U if and only if T is a subtype of U .

If EDT is the element type of L , then L can thus be considered as a function of the integers in the range 1 (one) to n into EDT .

If LT is some list type with element type EDT , then every value of LT is a list of EDT . If LT is some distinct list type with element type EDT , then every value of LT is a distinct list of EDT .

Identity: Let $L1$ and $L2$ be lists of EDT . $L1$ and $L2$ are identical if and only if $L1$ and $L2$ have the same cardinality n and if, for every i in the range 1 (one) $\leq i \leq n$, the element at ordinal position i in $L1$ is identical to the element at ordinal position i in $L2$.

Comparison: Let $L1$ and $L2$ be lists. $L1$ is less than or equal than $L2$ if and only if $L1$ has a smaller cardinality than $L2$ or $L1$ and $L2$ have the same cardinality and one of the following is true:

- All elements of $L1$ are less than their corresponding elements at the same position in $L2$.
- Some leading elements of $L1$ are less than their corresponding elements at the same position in $L2$ and all following elements in $L1$ are equal to their corresponding elements in $L2$ at the same position.
- All elements of $L1$ are equal to their corresponding elements at the same position in $L2$.

4.16.1.2.2 Regular lists

A *list* or an *array* may contain duplicate elements.

4.16.1.2.3 Distinct lists

A *distinct list* or a *distinct array* is a *list* that does not contain duplicate elements, i.e., no two elements of a distinct list are equal.

NOTE 53 — A distinct list may contain multiple null values.

4.16.1.3 Path type

« WG3:BER-081 »

A material value of *path type* is a path.

Strict elements: Every path is a strict element of the path type.

Subtyping: The path type is a subtype of the collection type whose element type is the graph element type.

Identity: Let $P1$ and $P2$ be paths. $P1$ and $P2$ are identical if and only if $P1$ and $P2$ have the same cardinality n and if, for every i in the range 1 (one) $\leq i \leq n$, the graph element at ordinal position i in $P1$ is identical to the graph element at ordinal position i in $P2$.

Comparison: Let $P1$ and $P2$ be paths. $P1$ is less than or equal than $P2$ if and only if $P1$ has a smaller cardinality than $P2$ or $P1$ and $P2$ have the same cardinality and one of the following is true:

- All graph elements of $P1$ are less than the corresponding graph element at the same position in $P2$.
- Some leading graph elements of $P1$ are less than the corresponding graph element at the same position in $P2$ and all following graph elements in $P1$ are equal to the corresponding graph element in $P2$ at the same position.

- All graph elements of P_1 are equal to the corresponding graph element at the same position in P_2 .

4.16.1.4 Map type

A value of the *map type* with key type KT and value type VT is called a *map*. A map is an unordered collection of (key, value) pairs that are called entries, such that each possible key appears at most once in the collection. All the keys have the same value type that is a predefined value type.

** Editor's Note (number 49) **

Limitation on keys to be revisited. See Possible Problem [GQL-098](#).

Type characteristics: The map type with key type KT and value type VT is an abstract value type if either KT or VT is an abstract value type.

Strict elements: A map M is a strict element of a map type with key type KT and value type VT if and only if KT is the join of the most specific types of keys of all map entries of M and VT is the join of the most specific types of values of all map entries M .

Subtyping: A map type with key type KT_1 and value type VT_1 is a subtype of a map type with key type KT_2 and value type VT_2 if and only if KT_1 is a subtype of KT_2 and VT_1 is a subtype of VT_2 .

NOTE 54 — Since a map is unordered, there is no ordinal position to reference individual elements of a map. Individual values in a map are accessed by a look-up of the key instead.

Comparison: Maps are not comparable.

** Editor's Note (number 50) **

Decide if maps should be made comparable by sorting on their fields first. See Possible Problem [GQL-174](#).

4.16.1.5 Record type

« [BER-019](#) »

« [WG3:BER-040R3](#) »

A material value of a *record type* is a *record*. A *record* has a set of fields, each field has a field name and a field value.

Every record type is described by a record data type descriptor. A record data type descriptor comprises:

- The name of the base type of all record types (RECORD DATA).
« [WG3:BER-040R3](#) »
- An indication of whether the type contains records with additional fields. If this indication is true, then the record type is open. Otherwise it is closed.
- If the record type is closed, then a (possibly empty) set of field types. Each field type is a pair comprising a field name and a field value type. The field name is an identifier and the field value type is a value type. The field name of each field type is unique within the record type.
- An indication of whether the type contains the null value.

« [WG3:BER-040R3](#) »

For every record type RT , a record R is a material value of RT , if and only if either RT is open or:

- RT is closed.
- R has the same number of fields as RT has field types.

- For every field type in RT with field name FN and field value type FVT , R has a field with field name FN and a field value that is of FVT .

« WG3:BER-040R3 »

GQL-implementation is permitted to regard certain <record type>s as equivalent, if they have the same indication regarding the inclusion of records with additional fields, optional set of field types, and indication regarding the inclusion of the null value, as permitted by the Syntax Rules of Subclause 20.2, “<value type>”. When two or more <record type>s are equivalent, the GQL-implementation chooses one of these equivalent <record type>s as the normal form representing that equivalence class of <record type>s.

A record R is assignable to a site of a record type RT if R is of RT .

« WG3:BER-040R3 »

In this document, a *unit record type* is the record type that has zero field types. The only material value of the unit record type is the record with zero fields. This value is called the *unit record*.

The amendment of a record type $RT1$ with record type $RT2$ is a new record type that comprises:

- Every field type $FT1$ of $RT1$ for which it holds that the field name of every field type $FT2$ in $RT2$ differs from the field name of $FT1$.
- Every field type of $RT2$.

NOTE 55 — The definition gives precedence to a field FR in R over a field FO in the original current working record if FR and FO have the same field name. This document generally aims to prevent this case from arising by other means such as syntax restrictions. If $RT1$ and $RT2$ are material, then RT is material.

Type characteristics: A record type is an abstract data type if and only if the value type of any of the field types of the record type is an abstract data type.

Strict elements: A record is a strict element of the record type with a set of field types that exactly are the field types of the fields of the record and whose value types are the most specific types of the corresponding fields of the actual record.

« WG3:BER-040R3 deleted one paragraph »

Comparison: A value of a record $R1$ is compared with a value of a record $R2$ by component-wise comparison of fields with the equivalent name in the order obtained by sorting the union of all field names of $R1$ and $R2$ in standard sort order. Missing fields in either $R1$ or $R2$ are assumed to be the null value.

** Editor's Note (number 51) **

Revisit sort order. See Possible Problem **GQL-174**.

« BER-019 Subclause deleted »

4.16.2 Reference value types

« WG3:BER-040R3 deleted one Editor's Note »

« WG3:BER-040R3 deleted two paragraphs »

« WG3:BER-040R3 »

A material value of a *reference value type* is a *reference value*. A reference value effectively represents a reference to some globally identifiable object (its referent) by opaquely encapsulating its referent's global object identifier. Reference values may only be assigned to certain dynamic sites (such as temporary or transient sites) and are not persisted as part of catalog objects.

Every reference value type is described by a reference value data type descriptor. A reference value data type descriptor comprises:

- The *reference base type name* of the reference value type. This name shall always end with the reserved word REFERENCE.
- The *object base type name* of the reference value type. This name is the name of the common base type of the object types of all possible referents of the reference values of the reference value type.

NOTE 56 — In this document, the reference base type name and the object base type name of a reference value type always have the same base type name prefix.
- The optional *refined object type* of all possible referents of the reference values of the reference value type. The name of the base type of this object type is the object base type name.
- An indication of whether the type contains the null value.

The GQL language supports the following kinds of reference values:

- Node reference values whose referents are nodes.
- EdgerefERENCE values whose referents are edges.

« WG3:BER-040R3 deleted three paragraphs »

4.16.3 Predefined value types

**** Editor's Note (number 52) ****

WG3:W05-020 suggests that, whilst this Subclause is not considered controversial and has already been discussed in [WG3:W03-034], there is still a need to establish consensus. See Possible Problem [GQL-099].

4.16.3.1 Boolean types

The material values of a *Boolean type* are the distinct truth values *True* or *False*. The truth value *Unknown* is represented by the null value.

This document does not make a distinction between the null value and the truth value *Unknown* that is the result of a GQL <predicate>, <search condition>, or <boolean value expression>; they may be used interchangeably to mean exactly the same value.

Every Boolean type is described by its Boolean data type descriptor. A Boolean data type descriptor comprises:

- The name of the base type of all Boolean types (BOOLEAN DATA).
- The preferred name of the Boolean type (implementation-defined (ID021) choice of BOOLEAN or BOOL).
- An indication of whether the Boolean type includes the null value (which is indistinguishable from the truth value *Unknown*).

« WG3:BER-040R3 »

A GQL-implementation regards certain <boolean type>s as equivalent, if they have the same indication regarding the inclusion of the null value, as permitted by the Syntax Rules of Subclause 20.2, “<value type>”. When two or more <boolean type>s are equivalent, the GQL-implementation chooses one of these equivalent <boolean type>s as the normal form representing that equivalence class of <boolean type>s.

All Boolean values and truth values are assignable to a site of at least the Boolean type. The values *True* and *False* may be assigned to any site having at least material Boolean type; assignment of *Unknown*, or the null value, is subject to the nullability characteristic of the target.

4.16.3.2 Character string types

4.16.3.2.1 Introduction to character strings

A material value of a *character string type* is a possibly-empty variable-length string (sequence) of characters drawn from the Universal Character Set repertoire specified by [The Unicode® Standard](#).

A character string has a length that is the number of characters in the sequence. The length is 0 (zero) or a positive integer. The maximum length of a character string is implementation-defined ([IL013](#)) but shall be greater than or equal to 16383 characters.

With two exceptions, a character string value expression is assignable only to sites of at least a character string type. If a store assignment would result in the loss of characters due to truncation, then an exception condition is raised.

« WG3:BER-040R3 »

If evaluation of a <cast specification> would result in the loss of characters due to truncation, then a warning condition is raised.

A GQL-implementation may assume that all character strings are normalized in one of Normalization Form C (NFC), Normalization Form D (NFD), Normalization Form KC (NFKC), or Normalization Form KD (NFKD), as specified by [The Unicode® Standard](#). <normalized predicate> can be used to verify the normalization form to which a particular character string conforms. Applications can also use <normalize function> to enforce a particular <normal form>. With the exception of <normalize function> and <normalized predicate>, the result of any operation on an unnormalized character string is implementation-defined ([IA003](#)).

Every character string type is described by its character string data type descriptor. A character string data type descriptor contains:

- The name of the base type of all character string types (STRING DATA).
- The preferred name of the character string type (implementation-defined ([ID023](#)) choice of STRING or VARCHAR).
- An indication of whether the type includes the null value.
- The maximum length in characters of the character string type.

The maximum length in characters of a character string type shall be a positive integer.

« WG3:BER-040R3 »

A GQL-implementation regards certain <character string type>s as equivalent, if they have the same maximum length and indication regarding the inclusion of the null value, as permitted by the Syntax Rules of [Subclause 20.2, “<value type>”](#). When two or more <character string type>s are equivalent, the GQL-implementation chooses one of these equivalent <character string type>s as the normal form representing that equivalence class of <character string type>s.

4.16.3.2.2 Collations

A GQL-implementation defines a collation for character strings.

The following collations are recognised:

- UCS_BASIC is a collation in which the ordering is determined entirely by the Unicode scalar values of the characters in the strings being sorted.

NOTE 57 — The Unicode scalar value of a character is its code point treated as an unsigned integer.

4.16 Value types

- UNICODE is the collation in which the ordering is determined by applying the Unicode Collation Algorithm with the Default Unicode Collation Element Table, as specified in [Unicode Technical Standard #10](#).
- Other collations provided by the GQL-implementation.

The collation supported by the GQL-implementation is implementation-defined ([ID022](#)).

4.16.3.3 Byte string types

A material value of a *byte string type* is a sequence of bytes that is called a *byte string*.

« Editorial: Added 2¹⁶-2= for clarity »

A byte string has a length that is the number of bytes in the sequence. The length is 0 (zero) or a positive integer. The maximum length of a byte string is implementation-defined ([IL014](#)) but shall be greater than or equal to $2^{16}-2=65534$ bytes.

Every byte string type is described by its byte string data type descriptor. A byte string data type descriptor comprises:

- The name of the base type of all byte string types (BINARY DATA).
- The preferred name of the byte string type (implementation-defined ([ID024](#)) choice of BINARY or BYTES for fixed-length byte string types and implementation-defined ([ID027](#)) choice of VARBINARY or BYTES for variable-length byte string types).
- An indication of whether the byte string type includes the null value.
- The minimum length in bytes of the byte string type.
- The maximum length in bytes of the byte string type.

The minimum length in bytes of a byte string type shall be a non-negative integer.

The maximum length in bytes of a byte string type shall be a positive integer.

« WG3:BER-040R3 »

A GQL-implementation regards certain <byte string type>s as equivalent, if they have the same maximum length and indication regarding the inclusion of the null value, as permitted by [Subclause 20.2](#), “<value type>”. When two or more <byte string type>s are equivalent, the GQL-implementation chooses one of these equivalent <byte string type>s as the normal form representing that equivalence class of <byte string type>s.

« WG3:BER-040R3 »

A byte string is assignable only to sites of at least byte string type. If a store assignment would result in the loss of bytes due to truncation, then an exception condition is raised.

4.16.3.4 Numeric types**4.16.3.4.1 Introduction to numbers**

GQL supports two classes of numeric data:

- Exact numeric data
- Approximate numeric data

Exact numeric data is either signed or unsigned. Signed numeric data is either non-negative (positive or zero) or negative. Unsigned numeric data is always non-negative. Approximate numeric data is always signed.

Exact numeric types are either of base 2 (binary) or base 10 (decimal) precision. Signed binary exact numeric types are two's complement integers. Decimal exact numeric types may specify a scale factor. Approximate numeric types are of base 2 (binary) precision and may specify a scale factor.

Every *numeric type* is described by a numeric data type descriptor. A numeric data type descriptor comprises the following characteristics:

- The name of the base type of the numeric type (INTEGER DATA for exact numeric types with binary precision, DECIMAL DATA for exact numeric types with decimal precision, and FLOAT DATA for approximate numeric types).
- The preferred name of the specific numeric type, which is the declared name of the normal form of the numeric type.
- An indication of whether the type includes the null value.
- The (implemented) precision of the numeric type.
- For an exact numeric type with decimal precision or an approximate numeric type, the (implemented) scale of the numeric type.
- An indication of whether the precision and the scale are expressed in binary or decimal terms.
- The explicit declared precision, if any, of the numeric type.
- For an exact numeric type with decimal precision or an approximate numeric type, the explicit declared scale, if any, of the numeric type.

If <precision> and <scale> are not specified explicitly, then the corresponding element of the descriptor effectively contains the null value.

« WG3:BER-040R3 »

A GQL-implementation is permitted to regard certain <exact numeric type>s as equivalent, if they have the same precision, scale, radix, and indication regarding the inclusion of the null value, as permitted by the Syntax Rules of Subclause 20.2, “<value type>”. When two or more <exact numeric type>s are equivalent, the GQL-implementation chooses one of these equivalent <exact numeric type>s as the normal form representing that equivalence class of <exact numeric type>s. The normal form determines the preferred name of the exact numeric type in the numeric data type descriptor.

« WG3:BER-040R3 »

Similarly, a GQL-implementation is permitted to regard certain <approximate numeric type>s as equivalent, if they have the same indication regarding the inclusion of the null value, as permitted by the Syntax Rules of Subclause 20.2, “<value type>”, in which case the GQL-implementation chooses a normal form to represent each equivalence class of <approximate numeric type> and the normal form determines the preferred name of the approximate numeric type in the numeric data type descriptor.

For every numeric type, the least material value is less than or equal to zero and the greatest material value is greater than zero.

4.16.3.4.2 Characteristics of numbers

« WG3:BER-067R1 »

An exact numeric type has a precision P and a scale S . P is a positive integer that determines the number of significant digits in a particular radix R , where R is either 2 or 10. S is a non-negative integer. Every value of a signed exact numeric type of scale S is of the form $n \times 10^{-S}$, where n is an integer such that $-R^P \leq n < R^P$. Every value of an unsigned exact numeric type of scale S is of the form $n \times 10^{-S}$, where n is an integer such that 0 (zero) $\leq n < 2 \times R^P$.

NOTE 58 — Not every value in that range is necessarily a value of the type in question.

An exact numeric value consists of either one or more decimal digits followed by an optional decimal point and zero or more decimal digits or a decimal point followed by one or more decimal digits.

**** Editor's Note (number 53) ****

Further alignment between the definition of approximate numbers and the requirements of ISO/IEC 60559:2020 such as support for positive and negative infinity as well as NaNs and similar issues is needed. See [Language Opportunity \[GQL-217\]](#).

Approximate numeric values consist of a mantissa and an exponent. The mantissa is a signed numeric value, and the exponent is a signed integer that specifies the magnitude of the mantissa. Approximate numeric values have a precision. The precision of approximate numeric values is a positive integer that specifies the number of significant binary digits in the mantissa.

« WG3:BER-085 »

In this document, the sign of the mantissa is not considered as a significant binary digit for the purpose of determining the precision of an approximate numeric value.

« WG3:BER-010 P00-USA-444 »

The scale of approximate numeric values is its exponent size, i.e., a signed integer that specifies the number of significant binary digits of the exponent. The value of an approximate numeric type is the mantissa multiplied by a factor determined by the exponent.

NOTE 59 — A GQL-implementation may choose an internal representation for approximate numeric values that implicitly encodes leading bits instead of physically storing them. Any such implied bits are still part of the mantissa of any value represented using such an encoding.

An *<exact numeric literal>* *ENL* consists of either an *<unsigned integer>*, a *<period>* followed by an *<unsigned decimal integer>*, or an *<unsigned decimal integer>* followed by a *<period>* and an optional *<unsigned decimal integer>*. The declared type of *ENL* is an exact numeric type.

An *<approximate numeric literal>* *ANL* consists of a *<mantissa>* that is an *<exact numeric literal>*, the letter 'E' or 'e', and an *<exponent>* that is a *<signed decimal integer>*. The declared type of *ANL* is an approximate numeric type. If *M* is the value of the *<mantissa>* and *E* is the value of the *<exponent>*, then *M * 10^E* is the *apparent value* of *ANL*. If the declared type of *ANL* is an approximate numeric type, then the actual value of *ANL* is approximately the apparent value of *ANL*, according to implementation-defined ([IA004](#)) rules.

« WG3:W21-010 P00-USA-438 »

A number is assignable only to sites of at least a numeric type. If an assignment of some number would result in a loss of its most significant digit, an exception condition is raised. If least significant digits are lost, implementation-defined ([IA005](#)) rounding or truncating occurs, with no exception condition being raised. The rules for arithmetic are specified in [Subclause 19.4, "<numeric value expression>"](#).

Whenever a numeric value is assigned to an exact numeric value site, an approximation of its value that preserves leading significant digits after rounding or truncating is represented in the declared type of the target. The value is converted to have the precision and scale of the target. The choice of whether to truncate or round is implementation-defined ([IA005](#)).

An approximation obtained by truncation of a numeric value *N* for an exact numeric type *T* is a value *V* in *T* such that *N* is not closer to zero than *V* and there is no value in *T* between *V* and *N*.

An approximation obtained by rounding of a numeric value *N* for an exact numeric type *T* is a value *V* in *T* such that the absolute value of the difference between *N* and the numeric value of *V* is not greater than half the absolute value of the difference between two successive numeric values in *T*. If there is more than one such value *V*, then it is implementation-defined ([IA006](#)) which one is taken.

All numeric values between the smallest and the largest value, inclusive, in a given exact numeric type have an approximation obtained by rounding or truncation for that type; it is implementation-defined ([IA007](#)) which other numeric values have such approximations.

« WG3:BER-067R1 »

An approximation obtained by truncation or rounding of a numeric value N for an approximate numeric type T is a value V in T such that there is no numeric value in T distinct from that of V that lies between the numeric value of V and N , inclusive. If there is more than one such value V then it is implementation-defined (IA008) which one is taken.

It is implementation-defined (ID025) which numeric values have approximations obtained by rounding or truncation for a given approximate numeric type.

Whenever a numeric value is assigned to an approximate numeric value site, an approximation of its value is represented in the declared type of the target. The value is converted to have the precision of the target.

Operations on numbers are performed according to the normal rules of arithmetic, within implementation-defined (ID026) limits, except as provided for in Subclause 19.4, “<numeric value expression>”.

4.16.3.4.3 Binary exact numeric types

The signed binary exact numeric types are:

« WG3:BER-067R1 »

- The *signed 8-bit integer type* SIGNED INTEGER8 (alternatively: INTEGER8, INT8) with precision 7 and with scale 0 (zero).
- The *signed 16-bit integer type* SIGNED INTEGER16 (alternatively: INTEGER16, INT16) with precision 15 and with scale 0 (zero).
- The *signed 32-bit integer type* SIGNED INTEGER32 (alternatively: INTEGER32, INT32) with precision 31 and with scale 0 (zero).
- The *signed 64-bit integer type* SIGNED INTEGER64 (alternatively: INTEGER64, INT64) with precision 63 and with scale 0 (zero).
- The *signed 128-bit integer type* SIGNED INTEGER128 (alternatively: INTEGER128, INT128) with precision 127 and with scale 0 (zero).
- The *signed 256-bit integer type* SIGNED INTEGER256 (alternatively: INTEGER256, INT256) with precision 255 and with scale 0 (zero).
- The *signed regular integer type* SIGNED INTEGER (alternatively: INTEGER, INT) with implementation-defined (ID028) precision greater than or equal to 31 and with scale zero (0).
- The *signed small integer type* SIGNED SMALL INTEGER (alternatively: SMALL INTEGER, SMALLINT) with implementation-defined (ID029) precision less than or equal to the precision of the signed regular integer type SIGNED INTEGER and with scale zero (0).
- The *signed big integer type* SIGNED BIG INTEGER (alternatively: BIG INTEGER, BIGINT) with implementation-defined (ID030) precision greater than or equal to the precision of the signed regular integer type SIGNED INTEGER and with scale zero (0).
- The *signed user-specified integer types* SIGNED INTEGER(p) (alternatively: INTEGER(p), INT(p)) with implementation-defined (ID031) precision greater than or equal to p and with scale zero (0).

The unsigned binary exact numeric types are:

« WG3:BER-067R1 »

- The *unsigned 8-bit integer type* UNSIGNED INTEGER8 (alternatively: UINT8) with precision 7 and with scale 0 (zero).
- The *unsigned 16-bit integer type* UNSIGNED INTEGER16 (alternatively: UINT16) with precision 15 and with scale 0 (zero).

- The *unsigned 32-bit integer type* UNSIGNED INTEGER32 (alternatively: UINT32) with precision 31 and with scale 0 (zero).
- The *unsigned 64-bit integer type* UNSIGNED INTEGER64 (alternatively: UINT64) with precision 63 and with scale 0 (zero).
- The *unsigned 128-bit integer type* UNSIGNED INTEGER128 (alternatively: UINT128) with binary precision 127 and with scale 0 (zero).
- The *unsigned 256-bit integer type* UNSIGNED INTEGER256 (alternatively: UINT256) with precision 255 and with scale 0 (zero).
- The *unsigned regular integer type* UNSIGNED INTEGER (alternatively: UINT) with the same precision as the regular signed integer type SIGNED INTEGER and with scale zero (0).
- The *unsigned small integer type* UNSIGNED SMALL INTEGER (alternatively: USMALLINT) with the same precision as the signed small integer type SIGNED SMALL INTEGER and with scale zero (0).
- The *unsigned big integer type* UNSIGNED BIG INTEGER (alternatively: UBIGINT) with the same precision as the signed big integer type SIGNED BIG INTEGER and with scale zero (0).
- The *unsigned user-specified integer types* UNSIGNED INTEGER(p) (alternatively: UINT(p)) with implementation-defined (ID033) precision greater than or equal to p and with scale zero (0).

4.16.3.4.4 Decimal exact numeric types

The decimal exact numeric types are:

« WG3:BER-085 »

- The *regular decimal exact numeric type* DECIMAL (alternatively: DEC) with implementation-defined (ID034) precision greater than or equal to the precision of the signed regular integer type and with scale 0 (zero).
- The *user-specified decimal exact numeric types* DECIMAL(p) (alternatively: DEC(p)) with implementation-defined (ID035) precision greater than or equal to p and with scale 0 (zero).
- The user-specified decimal exact numeric types DECIMAL(p, s) (alternatively: DEC(p, s)) with implementation-defined (ID036) precision greater than or equal to p and with implementation-defined (ID080) scale of s .

4.16.3.4.5 Approximate numeric types

The approximate numeric types are:

« WG3:BER-085 »

- The *16-bit approximate numeric type* FLOAT16 with precision 10 and with scale 5.
NOTE 60 — The material value space of the 16-bit approximate numeric type FLOAT16 is defined to be compatible with the binary16 interchange format of IEEE Std 754:2019.
- The *32-bit approximate numeric type* FLOAT32 with precision 23 and with scale 8.
NOTE 61 — The material value space of the 32-bit approximate numeric type FLOAT32 is defined to be compatible with the binary32 interchange format of IEEE Std 754:2019.
- The *64-bit approximate numeric type* FLOAT64 with precision 52 and with scale 11.
NOTE 62 — The material value space of the 64-bit approximate numeric type FLOAT64 is defined to be compatible with the binary64 interchange format of IEEE Std 754:2019.
- The *128-bit approximate numeric type* FLOAT128 with precision 112 and with scale 15.

NOTE 63 — The material value space of the 128-bit approximate numeric type FLOAT128 is defined to be compatible with the binary128 interchange format of IEEE Std 754:2019.

- The *256-bit approximate numeric type* FLOAT256 with precision 236 and with scale 19.
 NOTE 64 — The material value space of the 256-bit approximate numeric type FLOAT256 is defined to be compatible with the binary256 interchange format of IEEE Std 754:2019.
- The *regular approximate numeric type* FLOAT with implementation-defined (ID037) precision greater than or equal to 23 and with implementation-defined (ID038) scale greater than or equal to 8.
- The *real approximate numeric type* REAL with implementation-defined (ID039) precision less than or equal to the precision of the regular approximate numeric type FLOAT and with implementation-defined (ID040) scale.
- The *double approximate numeric type* DOUBLE (alternatively: DOUBLE PRECISION) with implementation-defined (ID041) precision greater than or equal to the precision of the regular approximate numeric type FLOAT and with implementation-defined (ID042) scale.
- The *user-specified approximate numeric types* FLOAT(*p*) with implementation-defined (ID043) binary precision greater than or equal to *p* and with implementation-defined (ID044) scale.
- The user-specified approximate numeric types FLOAT(*p, s*) with implementation-defined (ID046) precision greater than or equal to *p* and with implementation-defined (ID047) scale greater than or equal to *s*.

**** Editor's Note (number 54) ****

Need to classify into conformance features based on GQL-implementation limits (16 bit, 32 bit, 64 bit, 128 bit systems). See Possible Problem [GQL-198].

4.16.3.5 Temporal types

**** Editor's Note (number 55) ****

This Subclause requires further discussion. See Possible Problem [GQL-099].

**** Editor's Note (number 56) ****

Relationship to system-versioned graphs needs to be discussed. See Language Opportunity [GQL-185].

There are two classes of temporal types, temporal instant types and temporal duration types.

The time scale used for temporal instant types is defined in UTC-SLS.

The temporal instant types are:

- DATETIME

An instant capturing the date, the time, and the time zone.

NOTE 65 — Equivalent to TIMESTAMP WITH TIME ZONE with nanosecond precision in SQL.

- LOCALDATETIME

An instant capturing the date and the time, but not the time zone.

NOTE 66 — Equivalent to TIMESTAMP WITHOUT TIME ZONE with nanosecond precision in SQL.

- DATE

An instant capturing the date, but not the time, nor the time zone.

NOTE 67 — Equivalent to DATE in SQL.

— TIME

An instant capturing the time of day and the time zone, but not the date.

NOTE 68 — Equivalent to TIME WITH TIME ZONE with nanosecond precision in SQL.

— LOCALTIME

An instant capturing the time of day, but not the date, nor the time zone.

NOTE 69 — Equivalent to TIME WITHOUT TIME ZONE with nanosecond precision in SQL.

The temporal duration type is DURATION

A temporal amount. This captures the difference in time between two instants. It only captures the amount of time between two instants, it does not capture a start time and end time. A Duration can be negative.

A Duration captures time difference in a few different logical units. These units divide into three groups where conversion of units is possible within a group but not between groups (other than through applying the Duration to a point in time) as follows:

- Month-based units: months, quarters, years
- Day-based units: days, weeks
- Time-based units: hours, minutes, seconds, and subseconds (milliseconds, microseconds, nanoseconds)

NOTE 70 — This has some similarity to INTERVAL in SQL.

4.17 Sites

4.17.1 General description of sites

« BER-019 »

A *site* is a place occupied by an instance of a specified data type. Every site has a defined degree of persistence, independent of its data type. A site that exists until deliberately destroyed is said to be *persistent*. A site that necessarily ceases to exist on completion of a statement, at the end of a GQL-transaction, or at the end of a GQL-session is said to be *temporary*. A site that exists only for as long as necessary to hold a GQL-object or a value during the execution of an operation is said to be *transient*.

4.17.2 Assignment

The instance at a site can be changed by the operation of *assignment*. Assignment replaces the instance at a site with a new (possibly different) instance.

4.17.3 Nullability

« BER-019 »

Every site has a *nullability characteristic* that indicates whether the site may be occupied by the null value (is *possibly nullable*) or not (is *known not nullable*). In this document, the nullability characteristic of a site *S* and the indication whether the declared type of *S* includes the null value are always aligned.

5 Notation and conventions

5.1 Notation taken from The Unicode® Standard

The notation for the representation of UCS code points and sequences of code points is defined in [The Unicode® Standard](#), Appendix A, “Notational Conventions”.

In this document, this notation is used only to unambiguously identify characters and is not meant to imply a specific encoding for any GQL-implementation’s use of that character.

5.2 Notation

The syntactic notation used in this document is an extended version of BNF (“Backus Normal Form” or “Backus Naur Form”).

In a BNF language definition, each syntactic element, known as a *BNF non-terminal symbol*, of the language is defined by means of a *production rule*. This defines the element in terms of a formula consisting of the characters, character strings, and syntactic elements that can be used to form an instance of it.

In the version of BNF used in this document, the following symbols have the meanings shown in [Table 1, “Symbols used in BNF”](#).

Table 1 — Symbols used in BNF

Symbol	Meaning
< >	A character string enclosed in angle brackets is the name of a syntactic element (BNF non-terminal) of the GQL language.
: :=	The definition operator is used in a production rule to separate the element defined by the rule from its definition. The element being defined appears to the left of the operator and the formula that defines the element appears to the right.
[]	Square brackets indicate optional elements in a formula. The portion of the formula within the brackets may be explicitly specified or may be omitted.
{ }	Braces group elements in a formula. The portion of the formula within the braces shall be explicitly specified.
	The alternative operator. The vertical bar indicates that the portion of the formula following the bar is an alternative to the portion preceding the bar. If the vertical bar appears at a position where it is not enclosed in braces or square brackets, it specifies a complete alternative for the element defined by the production rule. If the vertical bar appears in a portion of a formula enclosed in braces or square brackets, it specifies alternatives for the content of the innermost pair of such braces or brackets.

Symbol	Meaning
...	The ellipsis indicates that the element to which it applies in a formula may be repeated any number of times. If the ellipsis appears immediately after a closing brace "}", then it applies to the portion of the formula enclosed between that closing brace and the corresponding opening brace "{". If an ellipsis appears after any other element, then it applies only to that element. In Syntax Rules, General Rules, and Conformance Rules, a reference to the n -th element in such a list assumes the order in which these are specified, unless otherwise stated.
!!	Introduces either a reference to the Syntax Rules, used when the definition of a syntactic element is not expressed in BNF, or the Unicode code point or code point sequence that define the character(s) of the BNF production.

Whitespace is used to separate syntactic elements. Multiple whitespace characters are treated as a single space. Apart from those symbols to which special functions were given above, other characters and character strings in a formula stand for themselves. In addition, if the symbols to the right of the definition operator in a production consist entirely of BNF symbols, then those symbols stand for themselves and do not take on their special meaning.

Pairs of braces and square brackets may be nested to any depth, and the alternative operator may appear at any depth within such a nest.

A character string that forms an instance of any syntactic element may be generated from the BNF definition of that syntactic element by application of the following steps:

- 1) Select any one option from those defined in the right hand side of a production rule for the element, and replace the element with this option.
- 2) Replace each ellipsis and the object to which it applies with one or more instances of that object.
- 3) For every portion of the character string enclosed in square brackets, either delete the brackets and their contents or change the brackets to braces.
- 4) For every portion of the character string enclosed in braces, apply Step 1) through Step 5) to the substring between the braces, then remove the braces.
- 5) Apply steps Step 1) through Step 5) to any BNF non-terminal symbol that remains in the character string.

The expansion or production is complete when no further non-terminal symbols remain in the character string.

The left normal form derivation of a character string CS in the source language character set from a BNF non-terminal NT is obtained by applying Step 1) through Step 5) above to NT , always selecting in Step 5) the leftmost BNF non-terminal.

5.3 Conventions

5.3.1 Specification of syntactic elements

Syntactic elements are specified in terms of:

- **Function:** A short statement of the purpose of the element.
- **Format:** A BNF definition of the syntax of the element.

- **Syntax Rules:** A specification in English of the syntactic properties of the element. These include rules for the following aspects that are specified in the sequence given:
 - The expansion of syntactic short-hand forms.
 - The normalization of syntactic forms.
 - Additional syntactic constraints, not expressed in BNF, that the element shall satisfy.
 - The visibility or scope of identifiers.
 - Specifying or defining the type of elements.

**** Editor's Note (number 57) ****

In GQL, both GRs and SRs may specify or define the type of elements currently. A deeper discussion of and further work on type checking in GQL in general is required to further develop this topic. See Possible Problem [GQL-007](#).

- **General Rules:** A specification in English of the run-time effect of the element. Where more than one General Rule is used to specify the effect of an element, the required effect is that which would be obtained by beginning with the first General Rule and applying the Rules in numeric sequence unless a Rule is applied that specifies or implies a change in sequence or termination of the application of the Rules. Unless otherwise specified or implied by a specific Rule that is applied, application of General Rules terminates when the last in the sequence has been applied.
- **Conformance Rules:** A specification of how the element shall be supported for conformance to GQL. Conformance Rules are effectively a kind of Syntax Rule, differentiated only because of their use to specify conformance to the GQL language. Conformance Rules in a given Subclause are therefore always applied concurrently with Syntax Rules of that Subclause.

The scope of notational symbols is the Subclause in which those symbols are defined. Within a Subclause, the symbols defined in Syntax Rules or General Rules can be referenced in other rules provided that they are defined before being referenced.

5.3.2 Specification of the Information Graph Schema

The objects of the Information Graph Schema in this document are specified in terms of:

- **Function:** A short statement of the purpose of the definition.
- **Definition:** A definition, in the GQL language, of the object being defined.
- **Description:** A specification of the run-time value of the object, to the extent that this is not clear from the definition.
- **Population:** A specification of the elements of the graph that a GQL-implementation shall provide that are not specified in the General Rules of other Subclauses.
- **Conformance Rules:** A specification of how the element shall be supported for conformance to GQL.

The only purpose of the Information Graph Schema is to provide access to the definitions contained in the schema. The actual objects on which the Information Graph Schema is based are implementation-dependent ([UV006](#)).

5.3.3 Use of terms

5.3.3.1 Syntactic containment

Let $\langle A \rangle$, $\langle B \rangle$, and $\langle C \rangle$ be syntactic elements; let A_1 , B_1 , and C_1 respectively be instances of $\langle A \rangle$, $\langle B \rangle$, and $\langle C \rangle$.

In a Format, $\langle A \rangle$ is said to *immediately contain* $\langle B \rangle$ if $\langle B \rangle$ appears on the right-hand side of the BNF production rule for $\langle A \rangle$. An $\langle A \rangle$ is said to *contain* or *specify* $\langle C \rangle$ if $\langle A \rangle$ immediately contains $\langle C \rangle$ or if $\langle A \rangle$ immediately contains a $\langle B \rangle$ that contains $\langle C \rangle$.

In GQL language, A_1 is said to *immediately contain* B_1 if $\langle A \rangle$ immediately contains $\langle B \rangle$ and B_1 is part of the text of A_1 . A_1 is said to *contain* or *specify* C_1 if A_1 immediately contains C_1 or if A_1 immediately contains B_1 and B_1 contains C_1 . If A_1 contains C_1 , then C_1 is *contained in* A_1 and C_1 is *specified by* A_1 .

A_1 is said to contain B_1 *with an intervening instance of* $\langle C \rangle$ if A_1 contains B_1 and A_1 contains an instance of $\langle C \rangle$ that contains B_1 . A_1 is said to contain B_1 *without an intervening instance of* $\langle C \rangle$ if A_1 contains B_1 and A_1 does not contain an instance of $\langle C \rangle$ that contains B_1 .

A_1 *simply contains* B_1 if A_1 contains B_1 without an intervening instance of $\langle A \rangle$ or an intervening instance of $\langle B \rangle$. If A_1 *simply contains* B_1 , then B_1 is *simply contained in* A_1 .

A_1 *directly contains* B_1 if A_1 contains B_1 without an intervening instance of \langle procedure body \rangle . If A_1 *directly contains* B_1 , then B_1 is *directly contained in* A_1 .

If an instance of $\langle A \rangle$ contains an instance of $\langle B \rangle$, then $\langle B \rangle$ is said to be *contained in* $\langle A \rangle$ and $\langle A \rangle$ is said to be a *containing* production symbol for $\langle B \rangle$. If an instance of $\langle A \rangle$ simply contains an instance of $\langle B \rangle$, then $\langle B \rangle$ is said to be *simply contained in* $\langle A \rangle$ and $\langle A \rangle$ is said to be a *simply containing* production symbol for $\langle B \rangle$. If an instance of $\langle A \rangle$ directly contains an instance of $\langle B \rangle$, then $\langle B \rangle$ is said to be *directly contained in* $\langle A \rangle$ and $\langle A \rangle$ is said to be a *directly containing* production symbol for $\langle B \rangle$.

A_1 is the *innermost* $\langle A \rangle$ satisfying a condition C if A_1 satisfies C and A_1 does not contain an instance of $\langle A \rangle$ that satisfies C . A_1 is the *outermost* $\langle A \rangle$ satisfying a condition C if A_1 satisfies C and A_1 is not contained in an instance of $\langle A \rangle$ that satisfies C .

In a Format, the verb “to be” (including all its grammatical variants, such as “is”) is defined as follows: $\langle A \rangle$ is said to be $\langle B \rangle$ if there exists a BNF production rule of the form $\langle A \rangle ::= \langle B \rangle$. If $\langle A \rangle$ is $\langle B \rangle$ and $\langle B \rangle$ is $\langle C \rangle$, then $\langle A \rangle$ is $\langle C \rangle$. If $\langle A \rangle$ is $\langle C \rangle$, then $\langle C \rangle$ is said to *constitute* $\langle A \rangle$. In GQL language, A_1 is said to be B_1 if $\langle A \rangle$ is $\langle B \rangle$ and the text of A_1 is the text of B_1 . Conversely, B_1 is said to *constitute* A_1 if A_1 is B_1 .

5.3.3.2 Terms denoting rule requirements

In the Syntax Rules, the term *shall* defines conditions that are required to be true of syntactically conforming GQL language. When such conditions depend on the contents or the structure of the GQL-catalog or the descriptors of GQL-objects reachable via the GQL-catalog, the GQL-session context, or the GQL-request context, they are required to be true just before the actions specified by the General Rules are performed. The treatment of language that does not conform to the Formats and Syntax Rules is implementation-defined (ID047). If any condition required by Syntax Rules is not satisfied when the evaluation of General Rules is attempted and the GQL-implementation is neither processing non-conforming GQL language nor processing conforming GQL language in a non-conforming manner, then an exception condition is raised as specified in Subclause 4.9.2, “Conditions”.

In the Conformance Rules, the term *shall* defines conditions that are required to be satisfied if the named Feature is or Features are not supported.

5.3.3.3 Rule evaluation order

« WG3:BER-026 deleted one Editor's Note »

A conforming GQL-implementation is not required to perform the exact sequence of actions defined in the General Rules, provided its effect on GQL-data and the GQL-catalog is identical to the effect of that sequence. The term *effectively* is used to emphasize actions whose effect may be achieved in other ways by a GQL-implementation.

The Syntax Rules and Conformance Rules for contained syntactic elements are effectively applied at the same time as the Syntax Rules and Conformance Rules for the containing syntactic elements. The General Rules for contained syntactic elements are effectively applied before the General Rules for the containing syntactic elements.

Where the precedence of operators is determined by the Formats of this document or by parentheses, those operators are effectively applied in the order specified by that precedence.

Where the precedence is not determined by the Formats or by parentheses, effective evaluation of expressions is *generally* performed from left to right. However, it is implementation-dependent (UA003) whether expressions are *actually* evaluated left to right, particularly when the evaluation of operands or operators causes conditions to be raised or if the results of the expressions may be determined without completely evaluating all parts of the expression.

If some syntactic element contains more than one other syntactic element, then the General Rules for contained elements that appear earlier in the production for the containing syntactic element are applied before the General Rules for contained elements that appear later.

For example, in the production:

```
<A> ::=  
    <B> <C>  
  
<B> ::=  
    ...  
  
<C> ::=  
    ...
```

the Syntax Rules and Conformance Rules for *<A>*, **, and *<C>* are effectively applied simultaneously. The General Rules for ** are applied before the General Rules for *<C>*, and the General Rules for *<A>* are applied after the General Rules for both ** and *<C>*.

An exception to this rule is when the General Rules of the containing syntactic element explicitly states when the General Rules of the contained syntactic element are to be applied.

NOTE 71 — In this document these exceptions are shown by the presence of a General Rule of the form “The General Rules of *xxx* are applied.” where *xxx* is a BNF non-terminal. This indicates that *xxx* is evaluated at this point and not at the point implied by the general “Rule evaluation order”.

If the result of an expression or search condition is not dependent on the result of some part of that expression or search condition, then that part of the expression or search condition is said to be *inessential*.

If evaluation of an inessential part would cause an exception condition to be raised, then it is implementation-dependent (UA004) whether or not that exception condition is raised.

During the computation of the result of an expression, the GQL-implementation may produce one or more *intermediate results* that are used in determining that result. The declared type of a site that contains an intermediate result is implementation-dependent (UV007).

5.3.3.4 Conditional rules

A conditional rule is specified with “If” or “Case” conventions. A rule specified with “Case” conventions includes a list of conditional subrules using “If” conventions. The first such “If” subrule whose condition is true is the effective subrule of the “Case” rule. The last subrule of a “Case” rule may specify “Otherwise”, in which case it is the effective subrule of the “Case” rule if no preceding “If” subrule in the “Case” rule is

satisfied. If the last subrule does not specify “Otherwise”, and if there is no subrule whose condition is true, then there is no effective subrule of the “Case” rule.

5.3.3.5 Syntactic substitution

In the Syntax and General Rules, the phrase “*X* is implicit” indicates that the Syntax and General Rules are to be interpreted as if the element *X* had actually been specified. Within the Syntax Rules of a given Subclause, it is known whether the element was explicitly specified or is implicit.

In the Syntax and General Rules, the phrase “the following <A> is implicit: *Y*” indicates that the Syntax and General Rules are to be interpreted as if a syntactic element <A> containing *Y* had actually been specified.

In the Syntax Rules and General Rules, the phrase “*former* is equivalent to *latter*” indicates that the Syntax Rules and General Rules are to be interpreted as if all instances of *former* in the element had been instances of *latter*.

If a BNF non-terminal is referenced in a Subclause without specifying how it is contained in a BNF production that the Subclause defines, then

Case:

- If the BNF non-terminal is itself defined in the Subclause, then the reference shall be assumed to be to the occurrence of that BNF non-terminal on the left side of the defining production.
- Otherwise, the reference shall be assumed to be to a BNF production in which the particular BNF non-terminal is immediately contained.

5.3.4 Descriptors

A *descriptor* is a coded description of a [GQL-object](#) that defines the *metadata* of a [GQL-object](#) of a specified type. The concept of descriptor is used in specifying the semantics of GQL. It is not necessary that any descriptor exist in any particular form in any GQL-environment.

Some GQL-objects cannot exist except in the context of other GQL-objects. For example, nodes cannot exist except within the context of graphs. Each such object is independently described by its own descriptor, and the descriptor of an enabling object (e.g., graph) is said to *include* the descriptor of each enabled object (e.g., node or edge). Conversely, the descriptor of an enabled object is said to *be included in* the descriptor of an enabling object.

[« WG3:BER-055 »](#)

The descriptor of some object *A* *generally includes* the descriptor of some object *C* if the descriptor of *A* includes the descriptor of *C* or if the descriptor of *A* includes the descriptor of *B* and the descriptor of *B* generally includes the descriptor of *C*.

[« Email from: Jan Michels 2022-06-16 »](#)

**** Editor's Note (number 58) ****

The definitions of “generally include” and “generally depend” are currently not used in the document. If they are still not used by the time the document exists the DIS stage, they should be removed. See [Possible Problem \[GQL-256\]](#).

In other cases, certain GQL-objects cannot exist unless some other GQL-object exists, even though there is no inclusion relationship. In general, a descriptor *D*₁ can be said to depend on, or to be dependent on, some descriptor *D*₂.

[« WG3:BER-055 »](#)

The descriptor of some object *A* *generally depends on* or *is generally dependent on* the descriptor of some object *C* if the descriptor of *A* depends on the descriptor of some object *C* or if the descriptor of *A* depends on the descriptor of some object *B* and the descriptor of *B* generally depends on the descriptor of *C*.

« Email from: Jan Michels 2022-06-16 »

**** Editor's Note (number 59) ****

The definitions of “generally include” and “generally depend” are currently not used in the document. If they are still not used by the time the document exists the DIS stage, they should be removed. See Possible Problem [GQL-256](#).

The execution of a statement may result in the creation of many descriptors. A GQL-object that is created as a result of a statement may depend on other descriptors that are only created as a result of the execution of that statement.

There are two ways of indicating dependency of one GQL-object on another. In many cases, the descriptor of the dependent GQL-object is said to “include the name of” the GQL-object on which it is dependent. In this case “the name of” is to be understood as meaning “sufficient information to identify the descriptor of”. Alternatively, the descriptor of the dependent GQL-object can be said to include text of the GQL-object on which it is dependent. However, in such cases, whether the GQL-implementation includes actual text (with defaults and implications made explicit) or its own style of parse tree is irrelevant; the validity of the descriptor is clearly “dependent on” the existence of descriptors of objects that are referenced in it.

An attempt to destroy a GQL-object, and hence its descriptor, may fail if other descriptors are dependent on it, depending on how the destruction is specified. Such an attempt may also fail if the descriptor to be destroyed is included in some other descriptor. Destruction of a descriptor results in the destruction of all descriptors included in it, but has no effect on descriptors on which it is dependent.

The implementation of some GQL-objects described by descriptors requires the existence of objects not specified by this document. Where such objects are required, they are effectively created whenever the associated descriptor is created and effectively destroyed whenever the associated descriptor is destroyed.

5.3.5 Subclauses used as subroutines

In this document, some Subclauses are defined without explicit syntax to invoke their semantics. Such Subclauses, called subroutine Subclauses, typically factor out rules that are required by one or more other Subclauses and are intended to be invoked by the rules of those other Subclauses.

In other words, the rules of these Subclauses behave as though they were a sort of definitional “subroutine” that is invoked by other Subclauses. These subroutine Subclauses are typically specified in a manner that requires information to be passed to them from their invokers. The information that is required to be passed is represented as parameters of these subroutine Subclauses, and that information is required to be passed in the form of arguments provided by the invokers of these subroutine Subclauses.

Every invocation of a subroutine Subclause shall explicitly provide information for every required parameter of the subroutine Subclause being invoked.

NOTE 72 — In this document the invocation will occur in the form “The xxx Rules of Subclause *n.n*, “aaaaaaa”, with ...”.

5.3.6 Index typography

In the Indexes to this document, the following conventions are used:

- An index entry in **boldface** indicates the page where the word, phrase, or BNF non-terminal is defined.
- An index entry in *italics* indicates a page where the BNF non-terminal is used in a Format.
- An index entry in neither boldface nor italics indicates a page where the word, phrase, or BNF non-terminal is not defined, but is used other than in a Format (for example, in a heading, Function, Syntax Rule, General Rule, Conformance Rule, Table, or other descriptive text).

5.3.7 Feature ID and Feature Name

Features are either *standard-defined features* or *implementation-defined features*.

Standard-defined features are defined in this document. Implementation-defined features are defined by GQL-implementations (see [Subclause 24.5.3, “Extensions and options”](#)).

Features are referenced by a Feature ID and by a Feature Name. A Feature ID value comprises a letter and three digits.

Feature IDs whose letter is “V” are reserved for implementation-defined features.

Standard-defined features are optional features that are defined by implicit or explicit Conformance Rules. An implicit Conformance Rule is any normative statement that begins “Without Feature *nnn*,”. The Feature ID of a standard-defined feature is stable and can be depended on to remain constant.

For convenience, all of the features defined in this document are collected together in a non-normative annex.

6 GQL-requests

6.1 <GQL-request>

Function

« BER-019 »

Specify a GQL-request.

Format

```
<GQL-request> ::=  
  <GQL-program>
```

Syntax Rules

- 1) Let RQ be the <GQL-request>.
- 2) Let P be the principal identified by the authorization identifier AI associated with the GQL-agent on whose behalf RQ was submitted.
- 3) Let RCX be the current request context.
- 4) Let PR be the specified <GQL-program>.
- 5) The scope clause of RQ is PR .
- 6) Let the GQL-schema WS be determined as follows

Case:

- a) If no GQL-session has been established for the GQL-client, then WS is the current home schema.
- b) Otherwise, WS is the current session schema.

- 7) The scope of WS comprises PR .

- 8) RQ declares WS as a working schema.

NOTE 73 — If no current home schema is set and no session is established for the GQL-client, no working schema is declared.

- 9) Let the graph WG be determined as follows

Case:

- a) If no GQL-session has been established for the GQL-client, then WG is the current home graph.
- b) Otherwise, WG is the current session graph.

- 10) The scope of WG comprises PR .

- 11) RQ declares WG as the working graph.

« BER-019 »

- 12) The declared type of incoming working record of *PR* is the material unit record type.
- 13) The declared type of incoming working table of *PR* is the material unit binding table type.

General Rules

- 1) Case:
 - a) If no GQL-session has been established for the GQL-client, then let *S* be a new GQL-session with an associated session context *SCX* that is initialized as follows:
 - i) Set the authorization identifier of *SCX* to *AI*.
 - ii) Set the principal of *SCX* to *P*.
 - iii) Set the time zone identifier of *SCX* to the implementation-defined ([ID048](#)) default time zone identifier.
 - iv) Set the session schema of *SCX* to the current home schema.
NOTE 74 — If no current home schema is set, no session schema is set to *SCX*.
 - v) Set the session graph of *SCX* to the current home graph.
 - vi) Set the session parameters of *SCX* to the implementation-defined ([ID049](#)) default session parameters.
NOTE 75 — The scope of the session parameters is defined in [4th paragraph of Subclause 4.10.4.6, "Scope of names"](#).
 - vii) Set the current transaction of *SCX* to no transaction.
 - viii) Set the request context of *SCX* to no request context.
 - ix) Set the termination flag of *SCX* to *False*.
 - b) Otherwise, *S* is the GQL-session that has already been established for the GQL-client and *SCX* is the session context associated with *S*.
- 2) Set the execution stack of *RCX* to a new empty execution stack.
- 3) Set the request outcome of *RCX* to a successful outcome with an omitted result.
- 4) Set the current request context to *RCX*.
- 5) Push a new execution context *CTX* onto the current execution stack, initialized as follows:
 - a) Set the working table of *CTX* to a new unit binding table.
 - b) Set the working record of *CTX* to a new empty record.
 - c) Set the execution outcome of *CTX* to a successful outcome with an omitted result.
NOTE 76 — *CTX* becomes the current execution context.
- 6) The General Rules of *PR* are applied.
- 7) Set the current request outcome to the execution outcome of *CTX*.
- 8) Pop the current execution context.
NOTE 77 — This leaves the execution stack empty.
- 9) If the current termination flag is set to *True* and the current transaction is active, then:
 - a) The following statement is implicitly executed:

ROLLBACK

- b) Set a failed outcome as the current request outcome.
- 10) Return the current request outcome to the GQL-client for delivery to the GQL-agent.
- 11) Set no current request context.
- 12) If the current termination flag is set to *True*, then close S by disassociating it from the GQL-client and destroying SCX .

Conformance Rules

None.

6.2 <GQL-program>

Function

Define a GQL-program.

Format

```
« WG3:BER-076 »

<GQL-program> ::=>
  <session activity>
  | <transaction activity>
  | <session close command>

<session activity> ::=>
  <session clear command> [ <session parameter command>... ]
  | <session parameter command>...

<session parameter command> ::=>
  <session set command>
  | <session remove command>

<transaction activity> ::=>
  <start transaction command>
  [ <procedure specification> [ <end transaction command> ] ]
  | <procedure specification> [ <end transaction command> ]
  | <end transaction command>
```

Syntax Rules

« WG3:BER-076 deleted one SR »

None.

General Rules

None.

Conformance Rules

None.

« WG3:BER-076 deleted one Subclause »

7 Session management

7.1 <session set command>

Function

Set values in the session context.

Format

```

<session set command> ::==
  SESSION SET {
    <session set schema clause>
  | <session set graph clause>
  | <session set time zone clause>
  | <session set parameter clause>
  }

<session set schema clause> ::==
  SCHEMA <schema reference>

<session set graph clause> ::==
  <graph resolution expression>

<session set time zone clause> ::==
  TIME ZONE <set time zone value>

<set time zone value> ::==
  <string value expression>

<session set parameter clause> ::==
  <session parameter>

<session parameter> ::==
  [ PARAMETER ] <parameter definition>

```

Syntax Rules

« WG3:BER-088R1 »

- 1) If the <session set schema clause> *SSSC* is specified, then let *S* be the schema identified by the <schema reference> immediately contained in *SSSC*.

General Rules

- 1) If the <session set schema clause> is specified, then set the current session schema to *S*.
- 2) If a <session set graph clause> *SSGC* is specified, then:
 - a) Let *GRE* be the <graph resolution expression> that is immediately contained in *SSGC*.
 - b) Set the current session graph to the result of *GRE*.

- 3) If the <string value expression> immediately contained in <set time zone value> does not conform to the representation specified in clause 4.3 “Time scale components and units” of ISO 8601-1:2019 with the optional addition of an IANA Time Zone Database time zone designator enclosed between a <left bracket> and a <right bracket> or, if both an IANA Time Zone Database time zone designator and a ISO 8601-1:2019 time shift specification are present and the IANA Time Zone Database time zone designator does not resolve to the same value as the ISO 8601-1:2019 time shift specification then an exception is raised: *data exception — invalid time zone displacement value (22009)*.

**** Editor's Note (number 60) ****

During the discussion of WG3:W08-014 (See WG3:W09-001) it was apparent that there was no consensus on the best way to specify a time zone. Should the standard allow both an explicit offset and an IANA time zone designator and should the combination also be allowed. This topic needs to be revisited to establish consensus. See Possible Problem [GQL-182](#).

- 4) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

7.2 <session remove command>

Function

Remove a session parameter from the current session context.

Format

```
<session remove command> ::=  
  [ SESSION ] REMOVE <parameter> [ IF EXISTS ]
```

Syntax Rules

- 1) Let *SRC* be the <session remove command>.
- 2) Let *PN* be the <parameter name> directly contained in *SRC*.
- 3) If IF EXISTS is not specified, then the current session context shall have a session parameter with parameter name *PN*.

General Rules

- 1) If the current session context has a session parameter *SP* with name *PN*, then remove *SP* from the current session context
- 2) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

7.3 <session clear command>

Function

Remove all session parameters from the current session context.

Format

```
<session clear command> ::=  
  [ SESSION ] CLEAR
```

Syntax Rules

None.

General Rules

- 1) Remove each session parameter in the current session context.
- 2) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

7.4 <session close command>

Function

Close the current session.

Format

```
<session close command> ::=  
[ SESSION ] CLOSE
```

Syntax Rules

None.

General Rules

- 1) Set the current termination flag to *True*.

Conformance Rules

None.

8 Transaction management

8.1 <start transaction command>

Function

Start a new GQL-transaction and set its characteristics.

Format

```
<start transaction command> ::=  
    START TRANSACTION [ <transaction characteristics> ]
```

Syntax Rules

- 1) If <transaction characteristics> is omitted, then READ WRITE is implicit.

General Rules

- 1) If a GQL-transaction is currently active, then an exception condition is raised: *invalid transaction state — active GQL-transaction (25G01)*.
- 2) A new GQL-transaction *TX* is initiated.
- 3) The current transaction is set to *TX*.

NOTE 78 — This determines *TX* as the currently active GQL-transaction associated with the GQL-session of the currently executing GQL-request

Conformance Rules

None.

8.2 <end transaction command>

Function

Terminate a GQL-transaction.

Format

```
<end transaction command> ::=  
    <commit command>  
  | <rollback command>
```

Syntax Rules

None.

General Rules

None.

Conformance Rules

None.

8.3 <transaction characteristics>

Function

Specify GQL-transaction characteristics.

Format

```

<transaction characteristics> ::= 
  <transaction mode> [ { <comma> <transaction mode> }... ]

<transaction mode> ::= 
  <transaction access mode>
  | <implementation-defined access mode>

<transaction access mode> ::= 
  READ ONLY
  | READ WRITE

<implementation-defined access mode> ::= 
  !! See the Syntax Rules.

```

Syntax Rules

- 1) Let TC be the <transaction characteristics>.
- 2) TC shall contain at most one <transaction access mode>.
- 3) If TC does not contain a <transaction access mode>, then READ WRITE is implicit.
- 4) The Format and Syntax Rules for <implementation-defined access mode> are implementation-defined ([ID054](#)).

General Rules

None.

Conformance Rules

None.

8.4 <rollback command>

Function

Terminate the currently active GQL-transaction with rollback.

Format

```
<rollback command> ::=  
    ROLLBACK
```

Syntax Rules

None.

General Rules

- 1) If the currently active GQL-transaction is part of an encompassing transaction that is controlled by an agent other than the GQL-agent and the <rollback command> is not being implicitly executed, then an exception condition is raised: *invalid transaction termination (2D000)*.
- 2) All changes to GQL-data or the GQL-catalog that were made by the currently active GQL-transaction are canceled.
- 3) The currently active GQL-transaction is terminated and the current transaction of the current session context is set to no transaction.

Conformance Rules

None.

8.5 <commit command>

Function

Terminate the currently active GQL-transaction with commit.

Format

```
<commit command> ::=  
    COMMIT
```

Syntax Rules

None.

General Rules

- 1) If the currently active GQL-transaction is part of an encompassing transaction that is controlled by an agent other than the GQL-agent, then an exception condition is raised: *invalid transaction termination (2D000)*.
- 2) Case:
 - a) If any enforced constraint is not satisfied, then any changes to GQL-data or the GQL-catalog that were made by the currently active GQL-transaction are canceled and an exception condition is raised: *transaction rollback — integrity constraint violation (40002)*.

**** Editor's Note (number 61) ****

This only becomes necessary once constraints are introduced. See Language Opportunity [\[GQL-011\]](#).
 - b) If any other error preventing commitment of the GQL-transaction has occurred, then any changes to GQL-data or the GQL-catalog that were made by the currently active GQL-transaction are canceled and an exception condition is raised: *transaction rollback (40000)* with an implementation-defined [\(ID055\)](#) subclass value.
 - c) Otherwise, any changes to GQL-data or the GQL-catalog that were made by the currently active GQL-transaction are eligible to be perceived by all subsequent GQL-transactions.
- 3) The currently active GQL-transaction is terminated and the current transaction of the current session context is set to no transaction.

Conformance Rules

None.

9 Procedures

**** Editor's Note (number 62) ****

WG3:W05-020 suggests that the text in this clause has not been considered by WG3 and that consensus agreement is required.
See Possible Problem [GQL-066](#).

9.1 <procedure specification>

Function

Define the procedural logic of a procedure.

Format

```
<nested procedure specification> ::=  
  <left brace> <procedure specification> <right brace>  
  
<procedure specification> ::=  
  <catalog-modifying procedure specification>  
  | <data-modifying procedure specification>  
  | <query specification>
```

**** Editor's Note (number 63) ****

Rules for the derivation of the procedure signature of a <procedure specification>, a <catalog-modifying procedure specification>, a <data-modifying procedure specification>, and a <query specification>, from their <procedure body> need to be specified. See Possible Problem [GQL-021](#).

```
<catalog-modifying procedure specification> ::=  
  <procedure body>  
  
<nested data-modifying procedure specification> ::=  
  <left brace> <data-modifying procedure specification> <right brace>  
  
<data-modifying procedure specification> ::=  
  <procedure body>
```

Syntax Rules

- 1) If the <procedure specification> *PS* is specified, then:
 - a) Let *PB* be the <procedure body> immediately contained in the <catalog-modifying procedure specification>, <data-modifying procedure specification>, or <query specification> immediately contained in *PS*.
 - b) Case:
 - i) If *PB* simply contains a <catalog-modifying statement>, then *PS* is a <catalog-modifying procedure specification>.

9.1 <procedure specification>

- ii) If PB simply contains a <data-modifying statement>, then PS is a <data-modifying procedure specification>.
 - iii) Otherwise, PS is a <query specification>.
- 2) If the <catalog-modifying procedure specification> CPS is specified, then every <statement> simply contained in CPS shall be a <catalog-modifying statement>.
 « Email from: Karl Schendel, 2022-06-13 »
- 3) If the <data-modifying procedure specification> DPS is specified, then every <statement> S simply contained in DPS shall be either a <data-modifying statement> or a <query statement>. At least one such S shall be a <data-modifying statement>.

General Rules

- 1) The outcome and result of the <nested procedure specification> NPS are the outcome and result, respectively, of the <procedure specification> immediately contained in NPS .
- 2) If a <procedure specification> PS is specified, then:
 - a) If PS is immediately contained in a <transaction activity>, then a new child execution context $CONTEXT$ is created.
 - b) If no GQL-transaction is active, then a new GQL-transaction TX is initiated and the current transaction is set to TX .

NOTE 79 — This determines TX as the currently active GQL-transaction associated with the GQL-session of the currently executing GQL-request.

« WG3:BER-022 »

- c) If PS is a <catalog-modifying procedure specification> and a <data-modifying procedure specification> has already occurred in the current GQL-transaction and the GQL-implementation does not support Feature GC01, “Catalog and data statement mixing”, then an exception is raised: *invalid transaction state — catalog and data statement mixing not supported (25G02)*.
 - d) If PS is a <data-modifying procedure specification> and a <catalog-modifying procedure specification> has alreadyFeature GC01, “Catalog and data statement mixing”, then an exception is raised: *invalid transaction state — catalog and data statement mixing not supported (25G02)*.
 - e) The outcome and result of PS are the outcome and result, respectively, of the <catalog-modifying procedure specification>, the <data-modifying procedure specification>, or the <query specification> immediately contained in PS .
 - f) If PS is immediately contained in a <transaction activity>, then:
 - i) Let $OUTCOME$ be the current execution outcome available at the end of the application of these General Rules in $CONTEXT$ before $CONTEXT$ is implicitly popped and destroyed.
 - ii) Set the current execution outcome to $OUTCOME$.
- 3) The outcome and result of a <catalog-modifying procedure specification> CPS are the outcome and result, respectively, of the <procedure body> immediately contained in CPS .
 - 4) The outcome and result of a <nested data-modifying procedure specification> $NDPS$ are the outcome and result, respectively, of the <data-modifying procedure specification> simply contained in $NDPS$.
 - 5) The outcome and result of a <data-modifying procedure specification> DPS are the outcome and result, respectively, of the <procedure body> immediately contained in DPS .

Conformance Rules

None.

9.2 <query specification>

Function

Define the procedural logic of a query.

Format

```
<nested query specification> ::=  
  <left brace> <query specification> <right brace>  
  
<query specification> ::=  
  <procedure body>
```

Syntax Rules

- 1) Let QS be the <query specification>.
- 2) Let PB be the <procedure body> immediately contained in QS .
- 3) Every <statement> simply contained in PB shall be a <query statement>.

General Rules

- 1) If a <nested query specification> NQS is specified, then the outcome and result of NQS are the outcome and result, respectively, of the <query specification> simply contained in NQS .
- 2) If a <query specification> QS is specified, then the outcome and result of QS are the outcome and result, respectively, of the <procedure body> immediately contained in QS .

Conformance Rules

None.

9.3 <procedure body>

Function

« BER-019 »

Specify the body of a procedure.

Format

```
<procedure body> ::=  
  [ <at schema clause> ]  
  [ <binding variable definition block> ]  
  <statement block>  
  « BER-019 BNF deleted »  
  
<statement block> ::=  
  <statement> [ <then statement>... ]  
  
<then statement> ::=  
  THEN [ <yield clause> ] <statement>
```

Syntax Rules

- 1) Let *PB* be the <procedure body>.
- 2) Let *SBLK* be the <statement block>.
- 3) If *SBLK* directly contains a <catalog-modifying statement>, then *SBLK* shall directly contain at most one <statement>.
- 4) If *SBLK* directly contains a <focused linear query statement> or a <focused linear data-modifying statement>, then *SBLK* shall not directly contain an <ambient linear query statement> or an <ambient linear data-modifying statement>.
- 5) If *SBLK* directly contains an <ambient linear query statement> or an <ambient linear data-modifying statement>, then *SBLK* shall not directly contain a <focused linear query statement> or a <focused linear data-modifying statement>.
- 6) If *SBLK* directly contains a <linear query statement> that is a <select statement>, then every <linear query statement> directly contained in *SBLK* shall be a <select statement>.
- 7) If *SBLK* directly contains a <linear query statement> that contains a <primitive result statement>, then every <linear query statement> directly contained in *SBLK* shall contain a <primitive result statement>.
- 8) If *SBLK* directly contains a <project statement>, then *SBLK* shall directly contain at most one <statement>.

« BER-019 »

- 9) If the <binding variable definition block> *BVDBLK* was specified, then
 - a) The declared type of the incoming working record of *BVDBLK* is the declared type of the incoming working record of *PB*.
 - b) The declared type of the incoming working table of *BVDBLK* is the material unit binding table type.

- 10) Let $TSTMSEQ$ be the sequence of <then statement>s immediately contained in $SBLK$, let m be the number of elements of $TSTMSEQ$.
- 11) Let the <statement>s STM_j , $0 \leq j \leq m$, directly contained in $SBLK$ be determined as follows:
 - a) STM_0 is the <statement> immediately contained in $SBLK$.
 - b) STM_j , $1 \leq j \leq m$, is the <statement> contained the j -th element of $TSTMSEQ$.
- 12) The declared type of incoming working record of STM_0 is defined as follows:

Case:

 - a) If the <binding variable definition block> $BVDBLK$ was specified, then the declared type of the incoming working record of STM_0 is the declared type of the outgoing working record of $BVDBLK$.
 - b) Otherwise, the declared type of incoming working record of STM_0 is the declared type of the incoming working record of PB .
- 13) The declared type of the incoming working table of STM_0 is the declared type of the incoming working table of PB .
- 14) For $1 \leq j \leq m$:
 - a) The declared type of the incoming working record of STM_j is the declared type of the outgoing working record of STM_{j-1} .
 - b) The declared type of the incoming working table of STM_j is defined as follows:

Case:

 - i) If the declared type of STM_{j-1} is a binding table type BTT , then

Case:

 - 1) If the <then statement> that directly contains STM_j also directly contains a <yield clause> YC , then
 - A) The declared type of the incoming working table of YC is BTT .
 - B) The declared type of the incoming working table of STM_j is the declared type of YC .
 - 2) Otherwise, the declared type of the incoming working table of STM_j is BTT .
 - ii) Otherwise, the declared type of the incoming working table of STM_j is the material unit binding table type.
- 15) The declared type of PB is the declared type of STM_m .

General Rules

**** Editor's Note (number 64) ****

All local variables defined by the <binding variable definition block> should be declared as known to be stable variables. This needs to be reflected in SRs. See Possible Problem **GQL-066**.

« BER-019 »

- 1) If the <binding variable definition block> *BVDBLK* is specified, then
 - a) the General Rules of *BVDBLK* are applied in a new child execution context *CONTEXT*.
 - b) Set the current working record to the working record of *CONTEXT*.
- 2) The General Rules of *STM*₀ are applied.
« BER-019 One rule deleted »
- 3) For 1 (one) $\leq j \leq m$:
 - a) Case:
 - i) If the current execution outcome has a result table *RT*, then set the current working table to *RT*.
 - ii) Otherwise, set the current working table to a new unit binding table.
« BER-019 »
 - b) If the <then statement> that directly contains *STM*_j also directly contains a <yield clause> *YC*, then the General Rules of *YC* are applied; set the current working table to be the *YIELD* returned from the application of these General Rules.
« BER-019 »
 - c) The General Rules of *STM*_j are applied.
« WG3:BER-099R1 »
- 4) The outcome of the application of these General Rules is the current execution outcome.

Conformance Rules

None.

10 Variable and parameter declaration and definition

** Editor's Note (number 65) **

WG3:W05-020 suggests that the text in this clause has not been considered by WG3 and that consensus agreement is required.
See Possible Problem [GQL-067](#).

10.1 Binding variable and parameter declaration and definition

Function

Declare and define variables.

Format

```

<compact variable declaration> ::= 
    <binding variable declaration> | <value variable>

<binding variable declaration> ::= 
    <graph variable declaration>
    | <binding table variable declaration>
    | <value variable declaration>

<compact variable definition> ::= 
    <compact value variable definition>
    | <binding variable definition>

<compact value variable definition> ::= 
    <value variable> <equals operator> <value expression>
    « BER-019 »

<binding variable definition block> ::= 
    <binding variable definition>...

<binding variable definition> ::= 
    <graph variable definition>
    | <binding table variable definition>
    | <value variable definition>

<parameter definition> ::= 
    <graph parameter definition>
    | <binding table parameter definition>
    | <value parameter definition>

```

Syntax Rules

- 1) If the <compact variable declaration> *CVD* is specified and *CVD* immediately contains the <value variable> *VV*, then *CVD* is effectively replaced by:

VALUE *VV*

- 2) If the <compact variable definition> *CVD* is specified and *CVD* immediately contains the <compact value variable definition> *CVVD*, then *CVD* is effectively replaced by:

10.1 Binding variable and parameter declaration and definition

VALUE CVVD

« BER-019 »

- 3) If the <binding variable definition block> *BVDBLK* was specified, then
 - a) Let *BVDSEQ* be the sequence of <binding variable definition> immediately contained in *BVDBLK*, let *m* be the number of elements of *BVDSEQ*.
 - b) The declared type of the incoming working record of the first <binding variable definition> in *BVDSEQ* is the declared type of the incoming working record of *BVDBLK*.
 - c) Let *BVD_j* be the *j*-th element of *BVDSEQ*, for $1 \leq j \leq m$.
 - d) The declared type of the incoming working record of *BVD_j* is the declared type of the outgoing working record of *BVD_{j-1}*, for $2 \leq j \leq m$.
 - e) The declared type of the outgoing working record of *BVDBLK* is the declared type of the outgoing working record of *BVD_j*.
 - f) The declared type of the outgoing working table of *BVDBLK* is the declared type of the incoming working table of *BVDBLK*.

General Rules

None.

« BER-019 One rule deleted »

Conformance Rules

None.

10.2 Graph variable and parameter declaration and definition

Function

« BER-019 »

Declare or define a graph variable.

Format

```

<graph variable declaration> ::==
  [ PROPERTY ] GRAPH <graph variable> <of graph type>
  « BER-019 »

<graph variable definition> ::==
  [ PROPERTY ] GRAPH <graph variable> [ <of graph type> ] <graph initializer>

<graph parameter definition> ::==
  [ PROPERTY ] GRAPH <parameter name> [ IF NOT EXISTS ]
  [ <of graph type> ] <graph initializer>

<graph variable> ::==
  <binding variable name>

<graph initializer> ::==
  <as or equals> <graph expression>
  | [ AS ] <nested graph query specification>
  | <colon> <catalog graph reference>

<as or equals> ::==
  AS | <equals operator>
  « BER-019 One note deleted »

```

Syntax Rules

- 1) If the <graph variable> *GV* is specified, then the name of *GV* is the <binding variable name> immediately contained in *GV*.
- 2) If the <graph initializer> *GI* is specified, then:

** Editor's Note (number 66) **

Consider harmonizing this with similar material in Subclause 10.4, "Value variable and parameter declaration and definition". See Possible Problem [GQL-254](#).

- a) If *GI* immediately contains a <graph expression> *GE*, then the declared type of *GI* is the declared type of *GE*.
- b) If *GI* immediately contains the <nested graph query specification> *NGS*, then:
 - i) *GI* is effectively replaced by:
AS PROPERTY GRAPH *NGS*
 - ii) The declared type of *GI* is the declared type of *NGS*.
- c) If *GI* immediately contains the <catalog graph reference> *CGR*, then:
 - i) *GI* is effectively replaced by:

10.2 Graph variable and parameter declaration and definition

AS PROPERTY GRAPH CGR

- ii) The declared type of *GI* is the graph type of the graph identified by *CGR*.

« BER-019 »

- 3) If the <graph variable declaration> *GVDECL* is specified, then *GVDECL* declares a variable whose name is the name of the <graph variable> immediately contained in *GVDECL* and whose declared type is the type of the <of graph type> immediately contained in *GVDECL*.
- 4) If the <graph variable definition> *GVD* is specified, then:
 - a) Let *WRT* be the declared type of the incoming working record of *GVD*.
 - b) Let *GN* be the name of the <graph variable> immediately contained in *GVD*.
 - c) *WRT* shall not have a field type whose field name is *GN*.
 - d) Let *GIT* be the declared type of the <graph initializer> *GI* immediately contained in *GVD*.
 - e) Let type *GT* be defined as follows:

Case:

 - i) If *GVD* immediately contains an <of graph type> *OGT*, then
 - 1) *GIT* shall be assignable to the type specified by *OGT*.
 - 2) *GT* is the type specified by *OGT*.
 - ii) Otherwise, *GT* is *GIT*.
 - f) The declared type of the outgoing working record is a record type comprising all the field types of *WRT* and one additional field type with field name *GN* and field value type *GT*.
- 5) If the <graph parameter definition> *GPD* is specified, then:
 - a) Let *GPN* be the <parameter name> immediately contained in *GPD*.
 - b) If IF NOT EXISTS is not specified, then the current session context shall not have a session parameter with parameter name *GPN*.
 - c) If the current session context does not have a session parameter with parameter name *GPN*, then:
 - i) Let *GT* be the declared type of the <graph initializer> *GI* immediately contained in *GPD*.
 - ii) Let the declared type of the session parameter with parameter name *GPN* be determined as follows

Case:

 - 1) If *GPD* immediately contains an <of graph type> *OGT*, then:
 - A) *GT* shall be assignable to the type specified by *OGT*.
 - B) The declared type of the session parameter with parameter name *GPN* is the type specified by *OGT*.
 - 2) Otherwise, the declared type of the session parameter with parameter name *GPN* is *GT*.

General Rules

« BER-019 One rule deleted »

- 1) If the <graph initializer> GI is specified, then the result of GI is the result of the <graph expression>, <nested graph query specification>, or <catalog graph reference> immediately contained in GI .
- 2) If the <graph variable definition> GVD is specified, then:

« BER-019 One rule deleted »

« BER-019 One rule deleted »

« BER-019 One rule deleted »

- a) Let $ROGI$ be the result of the <graph initializer> immediately contained in GVD .

« BER-019 »

- b) Add a field with field name GN and field value $ROGI$ to the current working record.

« BER-019 One rule deleted »

- 3) If the <graph parameter definition> GPD is specified, then:

- a) Let $ROPGI$ be the result of the <graph initializer> immediately contained in GPD .

- b) If the current session context does not have a session parameter with parameter name GPN , then add a session parameter with parameter name GPN and parameter value $ROPGI$ to the current session context.

Conformance Rules

None.

10.3 Binding table variable and parameter declaration and definition

Function

Declare or define a binding table variable.

Format

```

« WG3:BER-040R3 »

<binding table variable declaration> ::==
  [ BINDING ] TABLE <binding table variable> [ <of type> ] <binding table type>
« BER-019 »

<binding table variable definition> ::==
  [ BINDING ] TABLE <binding table variable> [ [ <of type> ] <binding table type> ]
    <binding table initializer>
« WG3:BER-040R3 »

<binding table parameter definition> ::==
  [ BINDING ] TABLE <parameter name> [ IF NOT EXISTS ]
    [ [ <of type> ] <binding table type> ] <binding table initializer>

<binding table variable> ::=
  <binding variable name>

<binding table initializer> ::=
  <as or equals> <binding table reference>
  | [ AS ] <nested query specification>
  | <colon> <catalog binding table reference>
« BER-019 One note deleted »

```

Syntax Rules

- 1) If the <binding table variable> *BTV* is specified, then the name of *BTV* is the <binding variable name> immediately contained in *BTV*.
- 2) If a <binding table initializer> *BTI* is specified, then the declared type of *BTI* is defined as follows:
Case:
 - a) If *BTI* immediately contains a <binding table reference> *BTR*, then the declared type of *BTI* is the binding table type of the binding table identified by *BTR*.
 - b) If *BTI* immediately contains a <nested query specification> *NQS*, then:
 - i) The declared type of *NQS* shall be a binding table type.
 - ii) The declared type of *BTI* is the declared type of *NQS*.
 - c) If *BTI* immediately contains a <catalog binding table reference> *CBTR*, then the declared type of *BTI* is the binding table type of the binding table identified by *CBTR*.

« BER-019 »

« WG3:BER-040R3 »

10.3 Binding table variable and parameter declaration and definition

- 3) If the <binding table variable declaration> *BTVDECL* is specified, then *BTVDECL* declares a variable whose name is the name of the <binding table variable> immediately contained in *BTVDECL* and whose declared type is the type of the <binding table type> immediately contained in *BTVDECL*.
- 4) If the <binding table variable definition> *BTVD* is specified, then:
 - a) Let *WRT* be the declared type of the incoming working record of *BTVD*.
 - b) Let *BTN* be the name of the <binding table variable> immediately contained in *BTVD*.
 - c) *WRT* shall not have a field type whose field name is *BTN*.
 - d) Let *BTIT* be the declared type of the <binding table initializer> *BTI* immediately contained in *BTVD*.
 - e) Let type *BTT* be defined as follows:

Case:

« WG3:BER-040R3 »

- i) If *BTVD* immediately contains a <binding table type> *OBTT*, then
 - 1) *BTIT* shall be assignable to the type specified by *OBTT*.
 - 2) *BTT* is the type specified by *OBTT*.
 - ii) Otherwise, *BTT* is *BTIT*.
 - f) The declared type of the outgoing working record is a record type comprising all the field types of *WRT* and one additional field type with field name *BTN* and field value type *BTT*.
 - 5) If the <binding table parameter definition> *BTPD* is specified, then:
 - a) Let *BTPN* be the <parameter name> immediately contained in *BTPD*.
 - b) If IF NOT EXISTS is not specified, then the current session context shall not have a session parameter with parameter name *BTPN*.
 - c) If the current session context does not have a session parameter with parameter name *BTPN*, then:
 - i) Let *BTT* be the declared type of the <binding table initializer> *BTI* immediately contained in *BTPD*.
 - ii) Let the declared type of the session parameter with parameter name *BTPN* be determined as follows
- Case:
- « WG3:BER-040R3 »
- 1) If *BTPD* immediately contains a <binding table type> *OBTT*, then:
 - A) *BTT* shall be assignable to the type specified by *OBTT*.
 - B) The declared type of the session parameter with parameter name *BTPN* is the type specified by *OBTT*.
 - 2) Otherwise, the declared type of the session parameter with parameter name *BTPN* is *BTT*.

General Rules

« BER-019 One rule deleted »

- 1) If the <binding table initializer> *BTI* is specified, then the result of *BTI* is the result of the <binding table reference>, <nested query specification>, or <catalog binding table reference> immediately contained in *BTI*.
- 2) If the <binding table variable definition> *BTVD* is specified, then:

« BER-019 Three rules deleted »

- a) Let *ROBTI* be the result of the <binding table initializer> immediately contained in *BTVD*.

« BER-019 »

- b) Add a field with field name *BTN* and field value *ROBTI* to the current working record.

« BER-019 One rule deleted »

- 3) If the <binding table parameter definition> *BTPD* is specified, then:

- a) Let *ROPBTI* be the result of the <binding table initializer> immediately contained in *BTPD*.

- b) If the current session context does not have a session parameter with parameter name *BTPN*, then add a session parameter with parameter name *BTPN* and parameter value *ROPBTI* to the current session context.

Conformance Rules

None.

10.4 Value variable and parameter declaration and definition

Function

Declare or define a value variable.

Format

```

« BER-019 »
« WG3:BER-040R3 »

<value variable declaration> ::==
  VALUE <value variable> [ <of type> ] <value type>

<value variable definition> ::==
  VALUE <value variable> [ [ <of type> ] <value type> ] <value initializer>
« WG3:BER-040R3 »

<value parameter definition> ::==
  VALUE <parameter name> [ IF NOT EXISTS ] [ [ <of type> ] <value type> ] <value initializer>

<value variable> ::=
  <binding variable name>

<value initializer> ::=
  <as or equals> <value expression>
  | [ AS ] <nested query specification>
  | <colon> <catalog object reference>

```

Syntax Rules

- 1) If the <value variable> *VV* is specified, then the name of *VV* is the <binding variable name> immediately contained in *VV*.
- 2) If a <value initializer> *VI* is specified, then the declared type of *VI* is defined as follows

**** Editor's Note (number 67) ****

Consider harmonizing this with similar material in Subclause 10.2, "Graph variable and parameter declaration and definition". See Possible Problem [GQL-254](#).

Case:

- a) If *VI* immediately contains a <value expression> *VE*, then the declared type of *VI* is the declared type of *VE*.
- b) If *VI* immediately contains a *nested query specification NQS*, then:

« WG3:BER-094R1 »

- i) The declared type of *NQS* shall be a predefined type, a list type, a map type, or a record type.
- ii) The declared type of *VI* is the declared type of *NQS*.
- c) If *VI* immediately contains a <catalog object reference> *COR*, then the declared type of *VI* is the declared type of the object identified by *COR*.

« BER-019 »

« WG3:BER-040R3 »

- 3) If the <value variable declaration> *VVDECL* is specified, then *VVDECL* declares a variable whose name is the name of the <value variable> immediately contained in *VVDECL* and whose declared type is the type of the <value type> immediately contained in *VVDECL*.
- 4) If the <value variable definition> *VVD* is specified, then:
 - a) Let *WRT* be the declared type of the incoming working record of *VVD*.
 - b) Let *VN* be the name of the <value variable> immediately contained in *VVD*.
 - c) *WRT* shall not have a field type whose field name is *VN*.
 - d) Let *VIT* be the declared type of the <value initializer> *VI* immediately contained in *VVD*.
 - e) Let type *VT* be defined as follows:
Case:
 i) If *VVD* immediately contains an <of graph type> *OVT*, then
 - 1) *VIT* shall be assignable to the type specified by *OVT*.
 - 2) *VT* is the type specified by *OVT*.
 ii) Otherwise, *VT* is *VIT*.
 - f) The declared type of the outgoing working record is a record type comprising all the field types of *WRT* and one additional field type with field name *VN* and field value type *VT*.
- 5) If the <value parameter definition> *VPD* is specified, then:
 - a) Let *VPN* be the <parameter name> immediately contained in *VPD*.
 - b) If IF NOT EXISTS is not specified, then the current session context shall not have a session parameter with parameter name *VPN*.
 - c) If the current session context does not have a session parameter with parameter name *VPN*, then:
 - i) Let *VT* be the declared type of the <value initializer> *VI* immediately contained in *VPD*.
 - ii) Let the declared type of the session parameter with parameter name *VPN* be determined as follows
Case:
 1) If *VPD* immediately contains a <value type> *OVT*, then:
 - A) *VT* shall be assignable to the type specified by *OVT*.
 - B) The declared type of the session parameter with parameter name *VPN* is the type specified by *OVT*.
 2) Otherwise, the declared type of the session parameter with parameter name *VPN* is *VT*.

General Rules

« BER-019 One rule deleted »

10.4 Value variable and parameter declaration and definition

- 1) If the <value initializer> VI is specified, then the result of VI is the result of the <value expression>, <nested query specification>, or <catalog object reference> immediately contained in VI .
- 2) If the <value variable definition> VVD is specified, then:
 « BER-019 Three rules deleted »
 - a) Let $ROVI$ be the result of the <value initializer> immediately contained in VVD .
 « BER-019 »
 - b) Add a field with field name VN and field value $ROVI$ to the current working record.
 « BER-019 One rule deleted »
- 3) If the <value parameter definition> VPD is specified, then:
 - a) Let $ROPVI$ be the result of the <value initializer> immediately contained in VPD .
 - b) If the current session context does not have a session parameter with parameter name VPN , then add a session parameter with parameter name VPN and parameter value $ROPVI$ to the current session context.

Conformance Rules

None.

11 Object and object type expressions

« WG3:BER-040R3 »

« WG3:BER-040R3 deleted one Editor's Note »

« BER-019 deleted one Subclause but BER-040 put it back »

11.1 Introduction to object and object type expressions

This clause provides the expressions and related BNF non-terminals needed for the definition of non-executable primary objects:

- Graphs (defined using a <graph expression>).
- Graph types (defined using a <graph type expression>).
- Binding table types (defined using a <binding table type>).

NOTE 80 — Binding tables are only defined by GQL-procedures returning them and therefore are not covered here.

« WG3:BER-040R3 deleted one Subclause »

11.2 <graph expression>

Function

Define a <graph expression>.

Format

```
« WG3:BER-039 »

<graph expression> ::= 
    <graph specification>
    | <graph reference>
« WG3:BER-039 moved one production »
```

Syntax Rules

None.

General Rules

- 1) Determine the graph G as follows

Case:

« WG3:BER-039 deleted one subrule »

- a) If GE simply contains the <graph specification> GS , then G is the graph defined by GS .
- b) If GE simply contains the <graph reference> GR , then G is the result of GR .

- 2) The result of GE is G and the graph descriptor identified by GE is the graph descriptor of G .

Conformance Rules

None.

11.3 <graph type expression>

Function

Define a <graph type expression>.

Format

```

<graph type expression> ::=

  <copy graph type expression>
  | <like graph expression>
  | <graph type specification>
  | <graph type reference>

<as graph type> ::=

  <as or equals> <graph type expression>
  | <like graph expression shorthand>
  | [ AS ] <nested graph type specification>

<copy graph type expression> ::=
  COPY OF <graph type reference>

<like graph expression> ::=
  [ PROPERTY ] GRAPH TYPE <like graph expression shorthand>
  « WG3:BER-040R3 »

<of graph type> ::=
  [ <of type> ] <graph type expression>
  | <like graph expression shorthand>
  | [ <of type> ] <nested graph type specification>

<like graph expression shorthand> ::=
  LIKE <graph expression>

```

** Editor's Note (number 68) **

Care needs to be taken to ensure that local graph references in a like graph expression shorthand are only resolved against the current working schema and otherwise ignore the execution stack. See Possible Problem [GQL-068](#).

Syntax Rules

- 1) If the <as graph type> *AGT* is specified and *AGT* immediately contains the <like graph expression shorthand> *LGES*, then *AGT* is effectively replaced by:

AS PROPERTY GRAPH TYPE *LGES*

- 2) If the <as graph type> *AGT* is specified and *AGT* immediately contains the <nested graph query specification> *NGQS*, then *AGT* is effectively replaced by:

AS PROPERTY GRAPH TYPE *NGQS*

- 3) If the <of graph type> *OGT* is specified and *OGT* immediately contains the <like graph expression shorthand> *LGES*, then *OGT* is effectively replaced by:

OF PROPERTY GRAPH TYPE *LGES*

- 4) If the <of graph type> *OGT* is specified and *OGT* immediately contains the <nested graph query specification> *NGQS*, then *OGT* is effectively replaced by:

```
OF PROPERTY GRAPH TYPE NGS
```

General Rules

- 1) Let *GTE* be the <graph type expression>.
- 2) The graph type *GT* is defined as follows

Case:

- a) If *GTE* simply contains the <copy graph type expression> that immediately contains the <graph type reference> *GTR*, then *GT* is a copy of the result of *GTR*.
 - b) If *GTE* simply contains the <like graph expression> that immediately contains the <like graph expression shorthand> that immediately contains the <graph expression> *GE*, then *GT* is the graph type of the result of *GE*.
 - c) If *GTE* simply contains the <graph type specification> *GTS*, then *GT* is the graph type defined by *GTS*.
 - d) If *GTE* simply contains the <graph type reference> *GTR*, then *GT* is the graph type of the result of *GTR*.
- 3) The result of *GTE* is *GT* and the graph type descriptor identified by *GTE* is the graph type descriptor of *GT*.

Conformance Rules

None.

« WG3:BER-040R3 »

11.4 <binding table type>

Function

Define a binding table type.

Format

```
« WG3:BER-040R3 deleted one production »  
  
<binding table type> ::=  
[ BINDING ] TABLE <field type specification>  
« WG3:BER-040R3 deleted two productions and one Editor's Note »
```

Syntax Rules

« WG3:BER-040R3 deleted one SR »

- 1) The base type of binding table types is named BINDING TABLE DATA.
- 2) Every <binding table type> *BTT* specifies a material binding table type whose record type is the closed material record type whose field types are specified by the <field type specification> immediately contained in *BTT*.
- 3) For each binding table type *BTT*, there is an implementation-dependent (UA007) binding table type, *BTNF(BTT)*, known as the normal form of *BTT* (which may be *BTT* itself), such that:
 - a) If *BTT1* and *BTT2* are two binding table types whose record types have the same normal form and whose indications regarding the inclusion of the null value are the same, then *BTNF(BTT1)* = *BTNF(BTT2)*.
 - b) *BTNF(BTNF(BTT))* = *BTNF(BTT)*.

General Rules

« WG3:BER-040R3 deleted four GRs »

- 1) If the <binding table type> specifies a binding table type *BTT*, then:
 - a) A record data type descriptor for the record type *RT* of *BTT* is created that comprises:
 - i) The base type name of all record types (RECORD DATA).
 - ii) An indication that *RT* does not include records with additional fields.
 - iii) A field type descriptor for every <field type> simply contained in *BTT*, according to the Syntax Rules and General Rules of Subclause 20.4, "<field type>", applied to the <field type>s in the order in which they were specified.
 - iv) An indication that *RT* includes the null value.
 - b) A binding table data type descriptor is created that comprises:
 - i) The base type name of all binding table types (BINDING TABLE DATA).
 - ii) The record data type descriptor of *RT*.

- iii) An indication that *BTT* excludes the null value.

Conformance Rules

None.

12 Statements

**** Editor's Note (number 69) ****

WG3:W05-020 suggests that the text in this clause, which was outlined in WG3:BNE-023, needs further discussion to achieve consensus agreement. See Possible Problem [GQL-069](#).

12.1 <statement>

Function

Define a <statement>.

Format

```
<statement> ::=  
  <catalog-modifying statement>  
  | <data-modifying statement>  
  | <query statement>  
  
<catalog-modifying statement> ::=  
  <linear catalog-modifying statement>  
  « WG3:BER-049 »  
  
<data-modifying statement> ::=  
  <linear data-modifying statement>  
  
<query statement> ::=  
  <composite query statement>
```

Syntax Rules

None.

General Rules

NOTE 81 — Not all statements set a material result; instead they may modify the current execution context. A material result is set by <primitive result statement>s and by <simple data-modifying statement>s to determine the execution outcome and result of a procedure or a top-level <statement>.

None.

Conformance Rules

None.

12.2 <call procedure statement>

Function

Define a <call procedure statement>.

Format

```
<call procedure statement> ::=  
  [ <statement mode> ] CALL <procedure call>  
  
<statement mode> ::=  
  OPTIONAL  
  | MANDATORY
```

**** Editor's Note (number 70) ****

Consider allowing <where clause>. See Language Opportunity [GQL-169](#).

**** Editor's Note (number 71) ****

Consider adding standalone calls. A standalone call is a syntax shorthand for a <call procedure statement> that implies `YIELD * RETURN *`

and that may only occur as valid singular (or perhaps last) top-level statement executed by a procedure. Standalone calls could be added by following existing syntactic precedence from Cypher, by extending the <project statement> or by introducing completely new syntax. See Language Opportunity [GQL-168](#).

Syntax Rules

**** Editor's Note (number 72) ****

BER-019 has established declared type propagation for working record, working table, and result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-257](#).

- 1) Let *CPS* be the <call procedure statement>.
- 2) Let *PC* be the <procedure call> immediately contained in *CPS*.
- 3) If *CPS* immediately contains a <statement mode> *SM*, then

Case:

- a) If *SM* is OPTIONAL, then *CPS* is effectively replaced by the <optional statement>:

```
OPTIONAL { CALL PC RETURN * }
```

- b) Otherwise, *CPS* is effectively replaced by the <mandatory statement>:

```
MANDATORY { CALL PC RETURN * }
```

General Rules

- 1) Let *TABLE* be the current working table.
- 2) Let *NEW_TABLE* be a new binding table.
- 3) For each record *R* of *TABLE* in a new child execution context amended with *R*:

- a) The General Rules of *PC* are applied; let *RESULT* be the binding table *RESULT* returned from the application of these General Rules.
 - b) Append the Cartesian Product of *R* and *RESULT* to *NEW_TABLE*.
- 4) Set the current working table to *NEW_TABLE*.
- 5) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

12.3 Statement classes

Function

Define various classes of statements.

Format

```
<simple catalog-modifying statement> ::=  
    <primitive catalog-modifying statement>  
    | <call catalog-modifying procedure statement>  
    « BER-020 »  
  
<primitive catalog-modifying statement> ::=  
    <create schema statement>  
    | <create graph statement>  
    | <create graph type statement>  
    | <drop schema statement>  
    | <drop graph statement>  
    | <drop graph type statement>  
  
<simple data-accessing statement> ::=  
    <simple query statement>  
    | <simple data-modifying statement>  
  
<simple data-modifying statement> ::=  
    <primitive data-modifying statement>  
    | <do statement>  
    | <call data-modifying procedure statement>  
    « WG3:BER-050 »  
  
<primitive data-modifying statement> ::=  
    <insert statement>  
    | <set statement>  
    | <remove statement>  
    | <delete statement>  
  
<simple query statement> ::=  
    <simple data-transforming statement>  
    | <simple data-reading statement>  
  
<simple data-reading statement> ::=  
    <match statement>  
    | <call query statement>  
  
<simple data-transforming statement> ::=  
    <primitive data-transforming statement>  
  
<primitive data-transforming statement> ::=  
    <optional statement>  
    | <mandatory statement>  
    | <let statement>  
    | <for statement>  
    | <aggregate statement>  
    | <filter statement>  
    | <order by and page statement>
```

Syntax Rules

None.

General Rules

None.

Conformance Rules

None.

13 Catalog-modifying statements

** Editor's Note (number 73) **

WG3:W05-020 suggests that this clause, which was accepted and adapted from [WG3:MMX-055] and [WG3:MMX-028r2], should be reviewed. See Possible Problem [GQL-147](#).

** Editor's Note (number 74) **

The GQL schema and metagraph need to be defined together with the statements to manipulate it. See Language Opportunity [GQL-004](#).

13.1 <linear catalog-modifying statement>

Function

« BER-019 »

Specify a linear composition of <simple catalog-modifying statement>s.

Format

```
<linear catalog-modifying statement> ::=  
  <simple catalog-modifying statement>...
```

Syntax Rules

« BER-019 »

- 1) Let $LCMS$ be the <linear catalog-modifying statement>.
- 2) Let $STMSEQ$ be the sequence of <simple catalog-modifying statement>s directly contained in $LCMS$, let m be the number of elements of $STMSEQ$ and let STM_j , $1 \leq j \leq m$, be the j -th element of $STMSEQ$.
- 3) Let $DTIWR$ be the declared type of the incoming working record of $LCMS$.
- 4) Let $DTIWT$ be the declared type of the incoming working table of $LCMS$.
- 5) For $1 \leq j \leq m$:
 - a) The declared type of the incoming working record of STM_j is $DTIWR$.
 - b) The declared type of the incoming working table of STM_j is $DTIWT$.
- 6) The declared type of the outgoing working record of $LCMS$ is $DTIWR$.
- 7) The declared type of the outgoing working table of $LCMS$ is $DTIWT$.
- 8) $LCMS$ has no declared type.

General Rules

None.

Conformance Rules

None.

13.2 <create schema statement>

Function

Create a schema.

Format

```
<create schema statement> ::=  
    CREATE SCHEMA <catalog schema parent and name> [ IF NOT EXISTS ]
```

Syntax Rules

- 1) After applying all relevant syntactic transformations, let *CSPN* be the <catalog schema parent and name>.
- 2) Let *SN* be the <schema name> immediately contained in *CSPN*.
- 3) Let *PARENT* be determined as follows

Case:

« WG3:BER-088R1 »

 - a) If *CSPN* does not immediately contain an <absolute url path>, then *PARENT* is the catalog root.
 - b) Otherwise, *PARENT* is the catalog object identified by the immediately contained <absolute url path>.
 - c) *PARENT* shall be a GQL-directory.
- 4) If IF NOT EXISTS is not specified, then *SN* shall not identify an existing schema descriptor in *PARENT*.

General Rules

- 1) If IF NOT EXISTS is specified and *SN* identifies an existing schema descriptor in *PARENT*, then no further General Rules of this Subclause are applied.
- 2) Let *S* be the GQL-schema described by the GQL-schema descriptor containing
 - a) *SN* as the name of the GQL-schema.
 - b) The current authorization identifier as the owner of the GQL-schema.
- 3) Add the descriptor of *S* to the GQL-directory descriptor of *PARENT*.

Conformance Rules

None.

13.3 <drop schema statement>

Function

Destroy a schema.

Format

```
<drop schema statement> ::=  
  DROP SCHEMA <catalog schema parent and name> [ IF EXISTS ]
```

Syntax Rules

- 1) After applying all relevant syntactic transformations, let *CSPN* be the <catalog schema parent and name>.
- 2) Let *SN* be the <schema name> immediately contained in *CSPN*.
- 3) Let *PARENT* be determined as follows

Case:

« WG3:BER-088R1 »

- a) If *CSPN* does not immediately contain an <absolute url path>, then *PARENT* is the catalog root.
 - b) Otherwise, *PARENT* is the catalog object identified by the immediately contained <absolute url path>.
 - c) *PARENT* shall be a GQL-directory.
- 4) If IF EXISTS is not specified, then *SN* shall identify an existing schema descriptor in *PARENT*.
 - 5) Let *SD* be the schema descriptor identified by *SN* in *PARENT*.
 - 6) *SD* shall not contain any catalog object descriptors.

General Rules

- 1) If IF EXISTS is specified and *SN* does not identify an existing schema descriptor in *PARENT*, then no further General Rules of this Subclause are applied.
- 2) If the schema of *SD* is the current working schema, an exception condition is raised:

** Editor's Note (number 75) **

Exception condition to be determined. See Possible Problem **GQL-158**.

- 3) Destroy *SD* in *PARENT*.

Conformance Rules

None.

13.4 <create graph statement>

Function

Create a graph.

Format

```
<create graph statement> ::=  
  CREATE {  
    [ PROPERTY ] GRAPH <catalog graph parent and name> [ IF NOT EXISTS ]  
    | OR REPLACE [ PROPERTY ] GRAPH <catalog graph parent and name>  
    } [ <of graph type> ] [ <graph source> ]  
  
<graph source> ::=  
  AS <copy graph expression>  
« WG3:BER-039 »  
  
<copy graph expression> ::=  
  COPY OF <graph expression>
```

Syntax Rules

- 1) Let *CGS* be the <create graph statement>.
- 2) After applying all relevant syntactic transformations, let *CGPN* be the <catalog graph parent and name> immediately contained in *CGS*.
- 3) Let *GPS* be the implicit or explicit <graph parent specification> immediately contained in *CGPN*.
- 4) Let *GN* be the <graph name> immediately contained in *CGPN*.
- 5) If *CGS* does not immediately contain a <graph source>, then the following <graph source> is implicit:
`AS COPY OF EMPTY_GRAPH`
- 6) Let *GSR* be the implicit or explicit <graph source> immediately contained in *CGS*.
- 7) After applying all relevant syntactic transformations, let *CGE* be the <copy graph expression> immediately contained in *GSR* and let *IGE* be the inner <graph expression> immediately contained in *CGE*.
- 8) If *CGS* does not immediately contain an <of graph type>, then *CGS* is replaced with:

```
CREATE PROPERTY GRAPH GPS GN LIKE IGE GSR
```

NOTE 82 — This rule rewrites:

```
CREATE PROPERTY GRAPH Y AS COPY OF X  
as
```

```
CREATE PROPERTY GRAPH Y LIKE X AS COPY OF X
```

i.e., it makes explicit that *X* provides the initial graph elements as well as the graph type for *Y*.

« BER-021 »

- 9) Let *PARENT* be the descriptor of the result of *GPS*.
- 10) If *CGS* does not immediately contain IF NOT EXISTS or OR REPLACE, then *GN* shall not identify an existing catalog object descriptor in *PARENT*.

- 11) If GN identifies an existing catalog object descriptor in $PARENT$, then that catalog object descriptor shall be a graph descriptor.

General Rules

« BER021 - deleted 3 rules »

- 1) If CGS immediately contains IF NOT EXISTS and GN identifies an existing graph descriptor in $PARENT$, then a completion condition is raised: *warning — graph already exists (01G01)* and no further General Rules of this Subclause are applied.
- 2) Let OGT be the <of graph type> immediately contained in CGS , let GTE be the <graph type expression> immediately contained in OGT , and let GTD be the graph type descriptor identified by GTE .
- 3) Let GTN be determined as follows

Case:

- a) If the graph type identified by GTD is not persistent, then:
 - i) Let $NGTN$ be a new system-generated name that does not identify an existing catalog object descriptor in $PARENT$.
 - ii) The following <create graph type statement> is effectively executed:

```
CREATE PROPERTY GRAPH TYPE GPS NGTN LIKE GE
```

- iii) GTN is $NGTN$.

- b) Otherwise, GTN is the graph type name of GTD .

- 4) Let GT be the graph type identified by GTN .

- 5) Let G be a graph with:

- a) GN as its graph name.
- b) An empty set as its graph label set.

**** Editor's Note (number 76) ****

The Format for a graph label set definition is not yet defined. See Possible Problem [GQL-253](#).

- c) An empty set as its graph property set.

**** Editor's Note (number 77) ****

The Format for a graph property set definition is not yet defined. See Possible Problem [GQL-253](#).

- d) GTN as the name of the graph type.

- 6) G is populated according to $GSRC$ as follows:

- a) Let SG be the graph specified by $GSRC$.
- b) If any graph element of SG does not conform to GT , then an exception condition is raised: *graph type conformance error (G2000)*.
- c) Copies of all nodes and edges in SG are inserted into G .

« WG3:BE-039 delete one Editor's Note »

13.4 <create graph statement>

- 7) If CGS immediately contains OR REPLACE and GN identifies an existing graph descriptor EG in $PARENT$, then the following <drop graph statement> is effectively executed:

```
DROP PROPERTY GRAPH GPS GN
```

- 8) A graph descriptor is created that describes G and includes:
- a) The name G of the graph.
 - b) The name GTN of the associated graph type.
- 9) G is created.

Conformance Rules

None.

13.5 <graph specification>

Function

Define a graph.

Format

```
<graph specification> ::=  

  [ PROPERTY ] GRAPH {  

    <nested graph query specification>  

  | <nested ambient data-modifying procedure specification>  

  }  
  

<nested graph query specification> ::=  

  <nested query specification>  
  

<nested ambient data-modifying procedure specification> ::=  

  <nested data-modifying procedure specification>
```

Syntax Rules

- 1) If the <nested ambient data-modifying procedure specification> *NADPS* is specified, then:
 - a) Let *NDPS* be the <nested data-modifying procedure specification> immediately contained in *NADPS*.
 - b) *NDPS* shall contain an <ambient linear data-modifying statement>.

General Rules

- 1) The graph defined by the <graph specification> *GS* is defined as follows
 Case:
 - a) If *GS* immediately contains the <nested graph query specification> *NGQS*, then the graph defined by *GS* is the result of *NGQS*.
 - b) If *GS* immediately contains the <nested ambient data-modifying procedure specification> *NADPS*, then the graph defined by *GS* is the result of *NADPS*.
- 2) If the <nested graph query specification> *NGQS* is specified, then:
 - a) Let *NQS* be the <nested query specification> immediately contained in *NGQS*.
 - b) Let *RONQS* be the result of *NQS*.
 - c) If *RONQS* is a graph *G*, then the graph defined by *NGQS* is *G*; otherwise, an exception condition is raised:

**** Editor's Note (number 78) ****

Exception condition to be determined. See Possible Problem **GQL-158**.
- 3) If the <nested ambient data-modifying procedure specification> *NADPS* is specified, then:

- a) Let $NDPS$ be the <nested data-modifying procedure specification> immediately contained in $NADPS$.
- b) The General Rules of $NADPS$ are applied in a new child execution context $CONTEXT$. Let G be the graph available as execution result in $CONTEXT$ after the application of these General Rules.
- c) Set the execution outcome to a successful outcome with result G .

Conformance Rules

None.

13.6 <drop graph statement>

« WG3:BER-037 deleted one Editor's Note »

Function

Destroy a graph.

Format

```
<drop graph statement> ::=  
  DROP GRAPH <catalog graph parent and name> [ IF EXISTS ]
```

Syntax Rules

- 1) After applying all relevant syntactic transformations, let *CGPN* be the <catalog graph parent and name>.
- 2) Let *GPS* be the implicit or explicit <graph parent specification> immediately contained in *CGPN*.
- 3) Let *GN* be the <graph name> immediately contained in *CGPN*.
« WG3:BER-037 »

** Editor's Note (number 79) **

It is not clear how <graph name> can be immediately contained in *CGPN* if *CGPN* immediately contains <url path parameter> and not <graph parent specification> <graph name>. See Possible Problem [GQL-300](#).

« WG3:BER-037 »

- 4) Let *PARENT* be the GQL-schema descriptor identified by *GPS*.
- 5) If IF EXISTS is not specified, then *GN* shall identify a graph descriptor *GD* included in *PARENT*.

General Rules

« WG3:BER-037 moved two GRs to become SRs »

- 1) If IF EXISTS is specified and *GN* does not identify an existing graph descriptor in *PARENT*, then a completion condition is raised: *warning — graph does not exist (01G03)* and no further General Rules of this Subclause are applied.
- 2) Let *GTN* be the graph type name contained in *GD*.
« WG3:BER-037 »
- 3) Destroy the graph descriptor identified by *GN* in *PARENT*.
« WG3:BER-037 deleted one Editor's Note »
- 4) If *GTN* is a system-generated name and is not contained in a graph type descriptor, then the following <drop graph type statement> is effectively executed:

```
DROP GRAPH TYPE GP GTN
```

Conformance Rules

None.

13.7 <create graph type statement>

Function

« BER-021 »

Create a graph type.

Format

```
<create graph type statement> ::=  
  CREATE {  
    [ PROPERTY ] GRAPH TYPE <catalog graph type parent and name> [ IF NOT EXISTS ]  
    | OR REPLACE [ PROPERTY ] GRAPH TYPE <catalog graph type parent and name>  
  } <graph type initializer>  
  
<graph type initializer> ::=  
  <as graph type>  
  | <colon> <catalog graph type reference>
```

Syntax Rules

- 1) Let *CGTS* be the <create graph type statement> and let *CGTPN* be the <catalog graph type parent and name> immediately contained in *CGTS*.
- 2) Let *GTPS* be the implicit or explicit <graph type parent specification> immediately contained in *CGTPN*.
- 3) Let *GTN* be the <graph type name> immediately contained in *CGTPN*.
« WG3:BER-088R1 deleted two SRs »
- 4) Let *GTI* be the <graph type initializer> immediately contained in *CGTS*.
« WG3:BER-088R1 »
- 5) If *GTI* is <colon> <catalog graph type reference>, then:
 - a) Let *CGTR* be the <catalog graph type reference> immediately contained in *GTI*.
 - b) *GTI* is equivalent to:

$$\text{AS PROPERTY GRAPH TYPE } \textit{CGTR}$$
- 6) Let *GTE* be the <graph type expression> immediately contained in *GTI*.
- 7) *GTE* shall not simply contain a <graph reference>. « BER-021 »
- 8) Let *PARENT* be the descriptor of the result of *GTPS*.
- 9) If *CGTS* does not immediately contain IF NOT EXISTS or OR REPLACE, then *GTN* shall not identify an existing catalog object descriptor in *PARENT*.
- 10) If *GTN* identifies an existing catalog object descriptor in *PARENT*, then that catalog object descriptor shall be a graph type descriptor.
- 11) If *CGTS* immediately contains OR REPLACE, then *GTN* shall not identify an existing graph type descriptor in *PARENT* that is identified in any graph descriptor.

General Rules

« BER-021 Deleted 2 Rules »

- 1) If *CTGS* immediately contains IF NOT EXISTS and *GTN* identifies an existing graph type descriptor in *PARENT*, then a completion condition is raised: *warning — graph type already exists (01G02)* and no further General Rules of this Subclause are applied.
- 2) Let *GTD* be the graph type descriptor defined by *GTE*.
- 3) If *CGTS* immediately contains OR REPLACE, then the following <drop graph type statement> is effectively executed:

```
DROP PROPERTY GRAPH TYPE GPS GTN
```

- 4) Create a graph type *GTN* in *PARENT* whose graph type descriptor is *GTD* with the name *GTN*.

Conformance Rules

None.

13.8 <graph type specification>

Function

Define a graph type.

Format

```

<graph type specification> ::==
  [ PROPERTY ] GRAPH TYPE <nested graph type specification>

<nested graph type specification> ::==
  <left brace> <graph type specification body> <right brace>

<graph type specification body> ::==
  <element type definition list>

<element type definition list> ::==
  <element type definition> [ { <comma> <element type definition> }... ]

<element type definition> ::==
  <node type definition>
  | <edge type definition>

```

Syntax Rules

None.

General Rules

- 1) The graph type defined by the <graph type specification> *GTS* is the graph type defined by the <graph type specification body> simply contained in *GTS*.
- 2) The graph type defined by the <nested graph type specification> *NGTS* is the graph type defined by the <graph type specification body> simply contained in *NGTS*.
- 3) The graph type defined by the <graph type specification body> *GTDB* is defined as follows:
 - a) Let *NTDS* be the set of <node type definition>s simply contained in *GTDB*.
 - b) Let *NTNS* be the set of <node type name>s simply contained in <node type definition>s contained in *NTDS*.
 - c) For every <node type definition> *NTD* contained in *NTDS* that simply contains a <node type name> *NTN*, let *NTDWN (NTN)* be *NTD*.
 - d) Let *ETDS* be the set of <edge type definition>s simply contained in *GTDB*.
 - e) Let *ETNS* be the set of <edge type name>s simply contained in <edge type definition>s contained in *ETDS*.
 - f) For every <edge type definition> *ETD* contained in *ETDS* that simply contains an <edge type name> *ETN*, let *ETDWN (ETN)* be *ETD*.
 - g) The base type of graph types is named GRAPH DATA.

« WG3:BER-040R3 »

h) The graph type defined by *GTDB* is described by the graph type descriptor containing:
« WG3:BER-040R3 »

- i) The name of the base type of all graph types (GRAPH DATA).

- ii) A graph type name that is empty.

NOTE 83 — The name will be supplied by the <create graph type statement> that contains it.

- iii) An empty set as the graph type label set.

NOTE 84 — The Format for a graph type label set definition is not yet defined.

- iv) An empty set as the graph type property type set.

NOTE 85 — The Format for a graph type property set definition is not yet defined.

- v) The set of all node type descriptors defined by the <node type definition>s contained in *NTDS* as the node type set.

- vi) The set of all edge type descriptors defined by the <edge type definition>s contained in *ETDS* as the edge type set.

- vii) The dictionary that maps every <node type name> *NTN* in *NTNS* to the node type defined by *NTDWN* (*NTN*) as the node type name dictionary.

- viii) The dictionary that maps every <node type name> *ETN* in *ETNS* to the edge type defined by *ETDWN* (*ETN*) as the edge type name dictionary.

Conformance Rules

None.

13.9 <node type definition>

Function

Define a node type.

Format

```
« WG3:BER-040R3 »

<node type definition> ::=

  <nnode type pattern>
  | <nnode synonym> <nnode type phrase>

<nnode type pattern> ::=

  <left paren> [ <nnode type name> ] [ <nnode type filler> ] <right paren>

<nnode type phrase> ::=

  [ TYPE ] <nnode type name> [ <nnode type filler> ]
  | <nnode type filler>

<nnode type name> ::=

  <element type name>

<nnode type filler> ::=

  <nnode type label set definition>
  | <nnode type property type set definition>
  | <nnode type label set definition> <nnode type property type set definition>

<nnode type label set definition> ::=

  <label set definition>

<nnode type property type set definition> ::=

  <property type set definition>
```

Syntax Rules

- 1) Let *NTD* be the <node type definition>.
 « WG3:BER-040R3 »
 - 2) If *NTD* is simply contained in a <refined node reference value type>, then *NTD* shall not contain both a <nnode type name> and a <nnode type filler>.
 - 3) If *NTD* is simply contained in a <graph type specification body> and contains a <nnode type name> *NTN* but does not contain a <nnode type label set definition>, then *NTD* is transformed as follows:
 - a) If *NTD* contains a <nnode type property type set definition>, then let *NTPTSD* be that <nnode type property type set definition>, otherwise let *NTPTSD* be the zero-length character string.
 - b) The following <nnode type filler> is implicit and replaces the existing <nnode type filler>, if any:

:*NTN NTPTSD*

NOTE 86 — This is node label implication.

« WG3:BER-040R3 »

- c) If *NTD* is simply contained in a <refined node reference value type>, then:

Case:

- i) If NTD contains a <node type name> NTN , then NTN shall identify a node type NT with name NTN in the graph type of the current working graph and the node type identified by NTD is NT .
- ii) Otherwise, NTD identifies the node type that it defines.

General Rules

- 1) If NTD contains a <node type label set definition> $NTLSD$, then let $NTLS$ the label set that is the result of the <label set definition> immediately contained in $NTLSD$. Otherwise let $NTLS$ be an empty label set.

« WG3:BER-040R3 »

- 2) The base type of node types is named NODE DATA. The preferred name of node types is implementation-defined ([ID087](#)) as either NODE or VERTEX.
- 3) If NTD contains a <node type property type set definition> $NTPTSD$, then let $NTPTS$ the property type set that is the result of the <property type set definition> immediately contained in $NTPTSD$. Otherwise let $NTPTS$ be an empty property type set.

« WG3:BER-040R3 »

- 4) If NTD is simply contained in a <graph type specification body> or contains no <node type name>, then the node type defined by NTD is described by a node type descriptor containing:
- a) The name of the base type of all node types (NODE DATA).
 - b) The node type name NTN , if specified.

**** Editor's Note (number 80) ****

The consequences of including the name need further consideration. See Possible Problem [GQL-038](#).

- c) The node type label set $NTLS$.
- d) The node type property type set $NTPTS$.

Conformance Rules

None.

13.10 <edge type definition>

Function

Define an edge type.

Format

```
« WG3:BER-040R3 »

<edge type definition> ::= 
    <edge type pattern>
    | [ <edge kind> ] <edge synonym> <edge type phrase>

<edge type pattern> ::= 
    <full edge type pattern>
    | <abbreviated edge type pattern>

<edge type phrase> ::= 
    [ TYPE ] <edge type name> [ <edge type filler> <endpoint definition> ]
    | <edge type filler> <endpoint definition>

<edge type name> ::= 
    <element type name>

<edge type filler> ::= 
    <edge type label set definition>
    | <edge type property type set definition>
    | <edge type label set definition> <edge type property type set definition>

<edge type label set definition> ::= 
    <label set definition>

<edge type property type set definition> ::= 
    <property type set definition>

<full edge type pattern> ::= 
    <full edge type pattern pointing right>
    | <full edge type pattern pointing left>
    | <full edge type pattern any direction>

<full edge type pattern pointing right> ::= 
    <source node type reference> <arc type pointing right> <destination node type reference>

<full edge type pattern pointing left> ::= 
    <destination node type reference> <arc type pointing left> <source node type reference>

<full edge type pattern any direction> ::= 
    <source node type reference> <arc type any direction> <destination node type reference>

<arc type pointing right> ::= 
    <minus left bracket> <arc type filler> <bracket right arrow>

<arc type pointing left> ::= 
    <left arrow bracket> <arc type filler> <right bracket minus>

<arc type any direction> ::= 
    <tilde left bracket> <arc type filler> <right bracket tilde>

<arc type filler> ::= 
    [ <edge type name> ] [ <edge type filler> ]
```

13.10 <edge type definition>

```

<abbreviated edge type pattern> ::= 
    <abbreviated edge type pattern pointing right>
    | <abbreviated edge type pattern pointing left>
    | <abbreviated edge type pattern any direction>

<abbreviated edge type pattern pointing right> ::= 
    <source node type reference> <right arrow> <destination node type reference>

<abbreviated edge type pattern pointing left> ::= 
    <destination node type reference> <left arrow> <source node type reference>

<abbreviated edge type pattern any direction> ::= 
    <source node type reference> <tilde> <destination node type reference>

« WG3:BER-040R3 »

<node type reference> ::= 
    <source node type reference>
    | <destination node type reference>

<source node type reference> ::= 
    <left paren> <source node type name> <right paren>
    | <left paren> [ <node type filler> ] <right paren>

<destination node type reference> ::= 
    <left paren> <destination node type name> <right paren>
    | <left paren> [ <node type filler> ] <right paren>

<edge kind> ::= 
    DIRECTED
    | UNDIRECTED

<endpoint definition> ::= 
    CONNECTING <endpoint pair definition>

<endpoint pair definition> ::= 
    <endpoint pair definition pointing right>
    | <endpoint pair definition pointing left>
    | <endpoint pair definition any direction>
    | <abbreviated edge type pattern>

<endpoint pair definition pointing right> ::= 
    <left paren> <source node type name> <connector pointing right> <destination node type name> <right paren>

<endpoint pair definition pointing left> ::= 
    <left paren> <destination node type name> <left arrow> <source node type name> <right paren>

<endpoint pair definition any direction> ::= 
    <left paren> <source node type name> <connector any direction> <destination node type name> <right paren>

<connector pointing right> ::= 
    TO
    | <right arrow>

<connector any direction> ::= 
    TO
    | <tilde>

<source node type name> ::= 
    <element type name>

<destination node type name> ::=

```

<element type name>

Syntax Rules

- 1) Let ETD be the <edge type definition>.
 « WG3:BER-040R3 »
 - 2) If ETD is simply contained in a <refined edge reference value type>, then:
 - a) ETD shall not contain both an <edge type name> and an <edge type filler>.
 - b) ETD shall not contain both an <edge type name> and an <edge kind>.
 - c) ETD shall not contain a <source node type name> or a <destination node type name>.
 - d) If ETD contains an <edge type name>, then ETD shall not contain a <node type filler>.
 - 3) If the <edge type phrase> ETP is simply contained in a <graph type specification body>, then ETP shall contain both an <edge kind> and an <edge type name>.
 - 4) If ETD immediately contains an <edge type name>, then let ETN be that <edge type name>, otherwise let ETN be the zero-length character string.
 - 5) Let $ETLSD$ be the <edge type label set definition> simply contained in ETD .
 - 6) If ETD contains an <edge type property type set definition>, then let $ETPTSD$ be that <edge type property type set definition>, otherwise let $ETPTSD$ be the zero-length character string.
 - 7) If ETD simply contains an <endpoint pair definition> EPD , then

Case:

 - a) If DIRECTED is simply contained in ETD , then EPD shall not contain an <endpoint pair definition any direction> or an <abbreviated edge type pattern any direction>.
 - b) If UNDIRECTED is simply contained in ETD , then EPD shall contain an <endpoint pair definition any direction> or an <abbreviated edge type pattern any direction>.
 « WG3:BER-040R3 »
 - 8) For each endpoint pair EP comprising two <node type reference>s, there is an implementation-dependent (UA008) endpoint pair, $EPPNF(EP)$, known as the normal form of EP (which may be EP itself), such that:
 - a) If NTR is a <node type reference>, then $EPPNF((NTR, NTR)) = (NTR, NTR)$.
 - b) If $NTR1$ and $NTR2$ are two <node type reference>s, then $EPPNF((NTR1, NTR2)) = EPPNF((NTR2, NTR1))$.
 - c) If $NTR1$ and $NTR2$ are two <node type reference>s, then $EPPNF((NTR1, NTR2))$ is either $(NTR2, NTR1)$ or it is $(NTR1, NTR2)$.
 - d) $EPPNF(EPPNF(EP)) = EPPNF(EP)$.
 - 9) ETD is transformed in the following steps:
 - a) If ETD simply contains an <endpoint pair definition> EPD and EPD does not contain an <abbreviated edge type pattern>, then:
 - i) Let $SNTN$ be the <source node type name> simply contained in EPD .
 - ii) Let $DNTN$ be the <destination node type name> simply contained in EPD .

13.10 <edge type definition>

iii) Case:

- 1) If DIRECTED is simply contained in *ETD*, then *EPD* is replaced by:

$(SNTN) \rightarrow (DNTN)$

- 2) If UNDIRECTED is simply contained in *ETD*, then *EPD* is replaced by:

$(SNTN) \sim (DNTN)$

NOTE 87 — This rewrites an *ETD* containing an <endpoint pair definition> that is not an <abbreviated edge type pattern> into an *ETD* containing an <endpoint pair definition> that is an <abbreviated edge type pattern>.

- b) If *ETD* simply contains an <abbreviated edge type pattern> *AETP*, then:

- i) Let *ASNTR* be the <source node type reference> simply contained in *AETP*.
- ii) Let *ADNTR* be the <destination node type reference> simply contained in *AETP*.
- iii) Case:

- 1) If *AETP* does not contain an <abbreviated edge type pattern any direction>, then *ETD* is replaced by:

$ASNTR - [ETN ETLSD ETPTSD] \rightarrow ADNTR$

« WG3:BER-040R3 »

- 2) If *AETP* contains an <abbreviated edge type pattern any direction>, then:

- A) Let $(ANSTR', ADNTR')$ be *EPPNF* $((ASNTR, ADNTR))$.
- B) *ETD* is replaced by:

$ASNTR \sim [ETN ETLSD ETPTSD] \sim ADNTR$

NOTE 88 — This rewrites an *ETD* containing an <abbreviated edge type pattern> into an *ETD* containing a <full edge type pattern>.

- c) If *ETD* simply contains a <full edge type pattern pointing left> *FETPPL*, then:

- i) Let *SNTR* be the <source node type reference> simply contained in *FETPPL*.
- ii) Let *DNTR* be the <destination node type reference> simply contained in *FETPPL*.
- iii) *ETD* is replaced by:

$SNTR - [ETN ETLSD ETPTSD] \rightarrow DNTR$

NOTE 89 — This rewrites an *ETD* containing a <full edge type pattern pointing left> into an *ETD* containing a <full edge type pattern pointing right>.

« WG3:BER-040R3 »

- d) If *ETD* is simply contained in a <graph type specification body> and simply contains an <edge type name> but does not simply contain an <edge type label set definition>, then the following <edge type filler> is implicit and replaces the existing <edge type filler>, if any:

: *ETN ETPTSD*

NOTE 90 — This is edge label implication.

« WG3:BER-040R3 »

- e) If *ETD* is simply contained in a <graph type specification body> and simply contains a <source node type name> *SNTN*, then:

- i) *GTDB* shall simply contain a <node type definition> *NTD* that simply contains a <node type name> that immediately contains *SNTN*.
- ii) If *NTD* contains a <node type filler>, then let *NTF* be that <node type filler>, otherwise let *NTF* be the zero-length character string.
- iii) *SNTN* is replaced by:

NTF

NOTE 91 — This rewrites an *ETD* containing a <source node type name> into an *ETD* containing the appropriate <node type filler>.

« WG3:BER-040R3 »

- f) If *ETD* is simply contained in a <graph type specification body> and simply contains a <destination node type name> *DNTN*, then:
 - i) *GTDB* shall simply contain a <node type definition> *NTD* that simply contains a <node type name> that immediately contains *DNTN*.
 - ii) If *NTD* contains a <node type filler>, then let *NTF* be that <node type filler>, otherwise let *NTF* be the zero-length character string.
 - iii) *DNTN* is replaced by:

NTF

NOTE 92 — This rewrites an *ETD* containing a <destination node type name> into an *ETD* containing the appropriate <node type filler>.

« WG3:BER-040R3 »

- 10) If *ETD* is simply contained in an <edge reference value type>, then:

Case:

- a) If *ETD* contains an <edge type name> *ETN*, then *ETN* shall identify an edge type *ET* with name *ETN* in the graph type of the current working graph and the edge type identified by *ETD* is *ET*.
- b) Otherwise, *ETD* identifies the edge type that it defines.

General Rules

- 1) Let *ETD* be the <edge type definition> after the transformations in the Syntax Rules.
« WG3:BER-040R3 deleted one GR »
- 2) Let *SNT* be the node type defined by *SNTR*.
- 3) Let *DNT* be the node type defined by *DNTR*.
« WG3:BER-040R3 »
- 4) If *ETD* is simply contained in a <graph type specification body> *GTSB*, then:
 - a) Let *NTS* be the set of all node types defined by the <node type definition>s simply contained in *GTSB*.
 - b) *SNT* or *DNT* are not contained in *NTS*, then an exception condition is raised: *graph type definition error — endpoint node type not defined (G3001)*.
- 5) If *ETD* contains an <edge type label set definition> *ETLSD*, then let *ETLS* the label set that is the result of the <label set definition> immediately contained in *ETLSD*. Otherwise let *ETLS* be an empty label set.

13.10 <edge type definition>

- 6) If ETD contains an <edge type property type set definition> $ETPTSD$, then let $ETPTS$ the property type set that is the result of the <property type set definition> immediately contained in $ETPTSD$. Otherwise let $ETPTS$ be an empty property type set.
- « WG3:BER-040R3 »
- 7) The base type of edge types is named EDGE DATA. The preferred name of edge types is implementation-defined ([ID088](#)) as either EDGE or RELATIONSHIP.
- 8) If ETD is simply contained in a <graph type specification body> or contains no <edge type name>, then the edge type defined by ETD is described by the edge type descriptor containing:
- The name of the base type of all edge types (EDGE DATA).
 - The edge type name ETN , if specified.

**** Editor's Note (number 81) ****The consequences of including the name need further consideration. See Possible Problem [GQL-038](#).

- The edge type label set $ETLS$.
- The edge type property type set $ETPTS$.
- Case:
 - If ETD does not simply contain a <full edge type pattern any direction>, then:
 - An indication that the edge type is directed.
 - SNT as the source node type of ETD .
 - DNT as the destination node type of ETD .
 - Otherwise:
 - An indication that the edge type is undirected.
 - A set of endpoint node types containing SNT and DNT .

Conformance Rules

- Without Feature GA10, “Undirected edge patterns”, conforming GQL language shall not contain an <edge type definition> that simply contains an <edge kind> that is DIRECTED, an <endpoint pair definition> that is an <endpoint pair definition any direction>, or a <full edge type pattern> that is a <full edge type pattern any direction>.

13.11 <label set definition>

Function

Define a label set.

Format

```
<label set definition> ::=  
  LABEL <label name>  
 | LABELS <label expression>  
 | <is label expression>
```

Syntax Rules

- 1) Let LSD be the <label set definition>.
- 2) LSD shall not contain a <label disjunction> nor a <label negation>.« WG3:BER-040R3 »
- 3) Every <label set definition> LSD specifies the *label name set* whose elements are the label names specified by the <label name>s that are simply contained in LSD .
- 4) For each label set definition LSD , there is an implementation-dependent (UA009) label set definition, $LSDNF(LSD)$, known as the *normal form* of LSD (which may be LSD itself), such that:
 - a) If $LSD1$ and $LSD2$ are two label set definitions with the same label name sets, then $LSDNF(LSD1) = LSDNF(LSD2)$.
 - b) $LSDNF(LSDNF(LSD1)) = LSDNF(LSD2)$.

General Rules

« WG3:BER-040R3 »

- 1) A label set descriptor is created for LSD that describes LSD and comprises the label names from the label name set of LSD .

Conformance Rules

None.

13.12 <property type set definition>

Function

Define a property type set.

Format

```
<property type set definition> ::=  
  <left brace> [ <property type definition list> ] <right brace>  
  
<property type definition list> ::=  
  <property type definition> [ { <comma> <property type definition> }... ]  
« WG3:BER-040R3 moved on production »
```

Syntax Rules

- 1) Let $PTSD$ be the <property type set definition>.
 « WG3:BER-040R3 deleted three SRs »
 « WG3:BER-040R3 »
- 2) Every <property type set definition> $PTSD$ specifies the *property type set* whose elements are the property types specified by the <property type definition>s that are simply contained in $PTSD$.
- 3) For each property type set definition $PTSD$, there is an implementation-dependent (UA010) property type set definition, $PTSDNF(PTSD)$, known as the *normal form* of $PTSD$ (which may be $PTSD$ itself), such that:
 - a) If $PTSD1$ and $PTSD2$ are two property type set definitions that specify the same property type sets, then $PTSDNF(PTSD1) = PTSDNF(PTSD2)$.
 - b) $PTSDNF(PTSDNF(PTSD1)) = PTSDNF(PTSD2)$.

General Rules

« WG3:BER-040R3 deleted two GRs »

« WG3:BER-040R3 »

- 1) A property type set descriptor is created for $PTSD$ that describes $PTSD$ and includes the property type descriptors of each property type of $PTSD$.

Conformance Rules

None.

« WG3:BER-040R3 »

13.13 <property type definition>

Function

Define a property type.

Format

```
<property type definition> ::=  
  <property name> [ <of type> ] <property value type>
```

Syntax Rules

- 1) Let *PTDL* be the <property type definition list> that simply contains a <property type definition> *PTD*.
- 2) The <property name> shall not be equivalent to the <property name> of any other <property type definition> simply contained in *PTDL*.
- 3) The <property name> specifies the property name of *PT*.
- 4) The <property value type> specifies the property value type of *PT*.

General Rules

- 1) A data type descriptor is created that describes the property value type of the property type being defined.
- 2) A property type descriptor is created that describes the property type being defined. The property type descriptor includes the following:
 - a) The property name.
 - b) The data type descriptor of the property value type.

Conformance Rules

- 1) Without Feature GD01, “Nested record types”, conforming language shall not contain a <property type definition> that simply contains a <record type>.

13.14 <drop graph type statement>

« WG3:BER-037 deleted one Editor's Note »

Function

Destroy a graph type.

Format

```
<drop graph type statement> ::=  
  DROP [ PROPERTY ] GRAPH TYPE <catalog graph type parent and name> [ IF EXISTS ]
```

Syntax Rules

- 1) After applying all relevant syntactic transformations, let *CGTPN* be the <catalog graph type parent and name>.
- 2) Let *GTPS* be the implicit or explicit <graph type parent specification> immediately contained in *CGTPN*.
- 3) Let *GTN* be the <graph type name> immediately contained in *CGTPN*.
 « WG3:BER-037 »

** Editor's Note (number 82) **

It is not clear how <graph type name> can be immediately contained in *CGTPN* if *CGTPN* immediately contains <url path parameter> and not <graph type parent specification> <graph type name>. See Possible Problem [GQL-300](#).

« WG3:BER-037 »

- 4) Let *PARENT* be the GQL-schema descriptor identified by *GTPS*.
- 5) If IF EXISTS is not specified, then *GTN* shall identify a graph type descriptor *GTD* included in *PARENT*.
- 6) *CGTPN* shall not be referenced by any graph descriptor.

** Editor's Note (number 83) **

What is the scope of this reference? The GQL-schema, the GQL-catlog, the world?

General Rules

« WG3:BER-037 moved a GR to become an SR »

- 1) If IF EXISTS is specified and *GTN* does not identify an existing graph type descriptor in *PARENT*, then a completion condition is raised: *warning — graph type does not exist (01G04)* and no further General Rules of this Subclause are applied.
 « WG3:BER-037 »
- 2) Destroy the graph type descriptor identified by *GTN* in *PARENT*.
 « WG3:BER-037 deleted one Editor's Note »

Conformance Rules

None.

13.15 <call catalog-modifying procedure statement>

Function

Define a <call catalog-modifying procedure statement>.

Format

```
<call catalog-modifying procedure statement> ::=  
  <call procedure statement>
```

Syntax Rules

None.

General Rules

- 1) Let $CCPS$ be the <call catalog-modifying procedure statement>, let CPS be the <call procedure statement> immediately contained in $CCPS$, and let PC be the <procedure call> immediately contained in CPS .
- 2) Case:
 - a) If PC is the <inline procedure call> that immediately contains the <nested procedure specification> that does not immediately contain the <catalog-modifying procedure specification>, then an exception condition is raised:

** Editor's Note (number 84) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

- b) If PC is the <named procedure call> that immediately contains the <procedure reference> PR and the result of PR is not a catalog-modifying procedure, then an exception condition is raised:

** Editor's Note (number 85) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

- 3) The General Rules of CPS are applied.

Conformance Rules

None.

14 Data-modifying statements

**** Editor's Note (number 86) ****

WG3:W05-020 suggests that, although the text in the Subclauses of this clause have been discussed in various papers, further discussion is needed to establish consensus agreement. See Possible Problem [GQL-071](#).

14.1 <linear data-modifying statement>

Function

« BER-019 »

Specify a linear composition of at least one <simple data-modifying statement> with <simple query statement>s and <simple data-accessing statement>s.

Format

```
<linear data-modifying statement> ::=  
  <focused linear data-modifying statement>  
  | <ambient linear data-modifying statement>  
  
<focused linear data-modifying statement> ::=  
  <focused linear data-modifying statement body>  
  | <focused nested data-modifying procedure specification>  
  
<focused linear data-modifying statement body> ::=  
  <use graph clause> [ <simple linear query statement> ]  
  <simple data-modifying statement>  
  [ <simple linear data-accessing statement> ]  
  [ <primitive result statement> ]  
  
<focused nested data-modifying procedure specification> ::=  
  <use graph clause> <nested data-modifying procedure specification>  
  « BER-019 »  
  
<ambient linear data-modifying statement> ::=  
  <ambient linear data-modifying statement body>  
  | <nested data-modifying procedure specification>  
  
<ambient linear data-modifying statement body> ::=  
  [ <simple linear query statement> ]  
  <simple data-modifying statement>  
  [ <simple linear data-accessing statement> ]  
  [ <primitive result statement> ]  
  
<simple linear data-accessing statement> ::=  
  <simple data-accessing statement>...
```

Syntax Rules

« BER-019 »

14.1 <linear data-modifying statement>

- 1) Let $LDMS$ be the <linear data-modifying statement>.
- 2) If the <focused linear data-modifying statement body> does not immediately contain a <primitive result statement>, END is implicit.
- 3) If the <ambient linear data-modifying statement body> does not immediately contain a <primitive result statement>, END is implicit.
- 4) Let $STMSEQ$ be the sequence of <simple query statement>s, <simple data-modifying statement>s, <simple data-accessing statement>s, the <primitive result statement>, and the <nested data-modifying procedure specification> directly contained in $LDMS$, let m be the number of elements of $STMSEQ$ and let STM_j , $1 \leq j \leq m$, be the j -th element of $STMSEQ$.
- 5) The declared type of the incoming working record of STM_1 is the declared type of the incoming working record of $LDMS$.
- 6) The declared type of the incoming working table of STM_1 is the declared type of the incoming working record of $LDMS$.
- 7) For $2 \leq j \leq m$:
 - a) The declared type of the incoming working record of STM_j is the declared type of the outgoing working record of STM_{j-1} .
 - b) The declared type of the incoming working table of STM_j is the declared type of the outgoing working record of STM_{j-1} .
- 8) The declared type of $LDMS$ is the declared type of STM_m .

General Rules

None.

Conformance Rules

None.

« WG3:BER-049 deleted one Subclause »

14.2 <do statement>

Function

Execute a nested procedure without modifying the current working table.

Format

```
<do statement> ::=  
    DO <nested data-modifying procedure specification>
```

** Editor's Note (number 87) **

Consider allowing a single optional <where clause>. See Language Opportunity [GQL-169](#).

Syntax Rules

** Editor's Note (number 88) **

BER-019 has established declared type propagation for working record, working table, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-259](#).

None.

General Rules

- 1) Let DS be the <do statement>.
- 2) Let $NDMPS$ be the <nested data-modifying procedure specification> immediately contained in DS .
- 3) For each record R of the current working table in a new child execution context amended with R , the General Rules of $NDMPS$ are applied.
- 4) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

14.3 <insert statement>

Function

Insert new nodes and edges into the current working graph.

**** Editor's Note (number 89) ****

Discussion paper WG3:MMX-047 suggests the addition of a "Time To Live" option, which would require specified graph elements be deleted after a certain time to save storage space. See Language Opportunity [GQL-035](#).

Format

```
<insert statement> ::=  
    INSERT <simple graph pattern>  
    | OPTIONAL INSERT <simple graph pattern> [ <when clause> ]  
    « Consequence of WG3:BER-049 »  
  
<when clause> ::=  
    WHEN <search condition>
```

**** Editor's Note (number 90) ****

Consider allowing a single optional <where clause>. See Language Opportunity [GQL-169](#).

Syntax Rules

**** Editor's Note (number 91) ****

BER-019 has established declared type propagation for working record, working table, and result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-260](#).

- 1) Let *IS* be the <insert statement>.
- 2) Let *SGP* be the <simple graph pattern> immediately contained in *IS*.
- 3) If *IS* immediately contains OPTIONAL, then:
 - a) Case:
 - i) If *IS* immediately contains a <when clause> *WC*, then let *SC* be the <search condition> immediately contained in *WC*. Let *FS* be the <filter statement>:


```
FILTER SC
```

 - ii) Otherwise, let *FS* be the zero-length character string.
 - b) *IS* is effectively replaced by the <optional statement>:


```
OPTIONAL {  
  FS  
  INSERT SGP  
  RETURN *  
}
```
 - 4) For each <element property specification> *EPS* simply contained in *SGP*:
 - a) Let *EPP* be the <element pattern predicate> that simply contains *EPS*.

- b) Let EPF be the <element pattern filler> that simply contains EPP .
- c) Let $EVAR$ be determined as follows

Case:

 - i) If EPF simply contains an <element variable declaration>, then $EVAR$ is the <identifier> contained in the <element variable declaration> simply contained in EPF .
 - ii) Otherwise, $EVAR$ is an implementation-dependent (UV008) <identifier> distinct from every element variable, subpath variable, and path variable contained in GP .
- d) Let $PECL$ be the <property key value pair list> simply contained in EPS .
- e) Let $NOPEC$ be the number of <property key value pair>s simply contained in $PECL$.
- f) Let $PEC_1, \dots, PEC_{NOPEC}$ be the <property key value pair>s simply contained in $PECL$.
- g) For every i , $1 \leq i \leq NOPEC$:
 - i) Let $PROP_i$ be the <property name> simply contained in PEC_i .
 - ii) Let VAL_i be the <value expression> simply contained in PEC_i .
 - iii) Let $SETPEC_i$ be a <set item> formed as:

$$EVAR.PROP_i = VAL_i$$
- h) Let $EPSET$ be a <set item list> formed through the concatenation of <set item>s: $SETPEC_1$ AND ... AND ... $SETPEC_{NOPEC}$.
- i) EPP is effectively replaced by the zero-length character string and the following <set statement> is effectively inserted after IS :

`SET EPSET`

General Rules

- 1) Let $TABLE$ be the current working table.
- 2) Let NEW_TABLE be a new empty binding table.
- 3) For each record R of $TABLE$, in a new child execution context amended with R :
 - a) Insert new graph elements as specified by SGP and construct a new record NR that extends R with additional fields corresponding to the newly inserted graph elements.

**** Editor's Note (number 92) ****

More details need to be supplied. See Possible Problem [GQL-165](#).
 - b) Append NR to NEW_TABLE .
- 4) Set the current working table to NEW_TABLE .
- 5) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

« WG3:BER-050 deleted one Subclause »

14.4 <set statement>

Function

Set graph element properties and labels.

Format

```

« WG3:BER-036 »

<set statement> ::=  
    SET <set item list>

<set item list> ::=  
    <set item> [ { <comma> <set item> }... ]

<set item> ::=  
    <set property item>  
    | <set all properties item>  
    | <set label item>

<set property item> ::=  
    <binding variable> <period> <property name> <equals operator> <value expression>
« WG3:BER-036R1 »

<set all properties item> ::=  
    <binding variable> <equals operator> <left brace> <property key value pair list> <right  
    brace>
« WG3:BER-036 »

<set label item> ::=  
    <binding variable> <is or colon> <label set specification>

<label set specification> ::=  
    <label name> [ { <label set delimiter> <label name> }... ]

<label set delimiter> ::=  
    <ampersand>  
    | <colon>

```

**** Editor's Note (number 93) ****

Consider allowing a single optional <where clause>. See Language Opportunity [GQL-169](#).

Syntax Rules

**** Editor's Note (number 94) ****

BER-019 has established declared type propagation for working record, working table, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-262](#).

- 1) Let *SS* be the <set statement>.
- 2) Let *SIL* be the <set item list> immediately contained in *SS*.
 « WG3:BER-036 deleted one SR »
 « WG3:BER-036 »

- 3) For every <set item> *SI* immediately contained in *SIL*.

Case:

- a) If *SI* immediately contains a <set property item> *SPI*, then:

- i) The declared type of <binding variable> immediately contained in *SPI* shall be a graph element type.
- ii) The declared type of <value expression> immediately contained in *SPI* shall be a supported property value type or a regular list type of supported property value types.

**** Editor's Note (number 95) ****

A corresponding type check for the dynamically type case needs to be added to the General Rules when dynamic types are sufficiently defined. See Possible Problem [GQL-007](#).

- b) If *SI* immediately contains a <set all properties item> *SAPI*, then:

- i) The declared type of <binding variable> immediately contained in *SAPI* shall be a graph element type.

« WG3:BER-036R1 »

- ii) For every <property key value pair> *PKVP* immediately contained in *SAPI*, the <value expression> immediately contained in *PKVP* shall be a supported property value type or a regular list type of supported property value types.

« WG3:BER-036 »

- c) If *SI* immediately contains a <set label item> *SLI*, the declared type of <binding variable> immediately contained in *SLI* shall be a graph element type.

- 4) The <binding variable> and <property name> in a <set property item> shall not be equivalent to the <binding variable> and <property name> of any other <set property item> of *SIL*.
- 5) The <binding variable> in the <set all properties item> shall not be equivalent to the <binding variable> of any other <set all properties item> or <set property item> of *SIL*.

General Rules

- 1) Let *TABLE* be the current working table.
- 2) Let *n* be the number of records of *TABLE* and let *m* be the number of <set item>s immediately contained in *SIL*.
- 3) For each *i*-th record *R_i*, $1 \leq i \leq n$, of *TABLE*, in a new child execution context amended with *R_i*:

- a) For each *j*-th <set item> *SI_j*, $1 \leq j \leq m$, immediately contained in *SIL* determine the graph element *GE_{i,j}* as follows

Case:

- i) If *SI_j* immediately contains a <set property item> *SPI_j*, let *GE_{i,j}* be the value of the <binding variable> immediately contained in *SPI_j*.
- ii) If *SI_j* immediately contains a <set all properties item> *SAPI_j*, let *GE_{i,j}* be the value of the <binding variable> immediately contained in *SAPI_j*.

- iii) If SI_j immediately contains a <set label item> SLI_j , let $GE_{i,j}$ be the value of the <binding variable> immediately contained in SLI_j .

« WG3:BER-036 deleted one subrule »

- b) For each j -th <set item> SI_j , $1 \leq j \leq m$, immediately contained in SIL determine the property value $PV_{i,j}$ as follows

Case:

- i) If SI_j immediately contains a <set property item> SPI_j , let $PV_{i,j}$ be the result of the <value expression> immediately contained in SPI_j .

« WG3:BER-036R1 »

- ii) If SI_j immediately contains a <set all properties item> $SAPI_j$

- 1) Let p be the number of <property key value pair> immediately contained in $SAPI_j$.

- 2) For each k -th <property key value pair> $PKVP_k$ immediately contained in $SAPI_j$, $1 \leq k \leq p$, let $PV_{i,j,k}$ be the result of evaluating the <value expression> immediately contained in $PKVP_k$.

- iii) Otherwise, let $PV_{i,j}$ be undefined.

- c) For each j -th <set item> SI_j , $1 \leq j \leq m$, immediately contained in SIL if SI_j immediately contains a <set label item> SLI_j , then let the label names $LN_{i,j}$ be the collection of all <label name>s simply contained in SLI_j ; otherwise, let $LN_{i,j}$ be undefined.

« WG3:BER-036 »

- 4) If an equivalent pair $PNGE_{i,j}$ of property name $PN_{i,j}$ and graph element $GE_{i,j}$ appears more than once in the current working table, then it is implementation-defined (IA017) which one of the following occurs:

- a) An exception condition condition is raised: *data exception — multiple assignments to a graph element property (22GOM)*.
- b) An implementation-dependent (UV009) one of the equivalent pairs $PNGE_{i,j}$ is chosen and the corresponding $PV_{i,j}$ is used perform the assignment to $GE_{i,j}$.

- 5) For each i -th record R_i , $1 \leq i \leq n$, of $TABLE$ (using the same order as General Rule 3)), perform the following data-modifying operations in a new child execution context amended with R_i for each j -th <set item> SI_j , $1 \leq j \leq m$, immediately contained in SIL .

« WG3:BER-036 »

If $GE_{i,j}$ is not the null value, then:

Case:

- a) If SI_j immediately contains a <set property item> SPI_j , then:

- i) Let PN_j be the <property name> immediately contained in SPI_j .

- ii) Set the property PN_j of $GE_{i,j}$ to $PV_{i,j}$.

- b) If SI_j immediately contains a <set all properties item>, then:

« WG3:BER-036R1 »

- i) Remove all properties of GE_{ij} .
 - ii) Let p be the number of <property key value pair> immediately contained in $SAPI_j$.
 - iii) For each <property key value pair> $PKVP_k$ immediately contained in $SAPI_j$:
 - 1) Let PN be the <property name> immediately contained in $PKVP_k$.
 - 2) Set the property PN of GE_{ij} to $PV_{i,j,k}$.
 - c) If SI_j immediately contains a <set label item>, then, for each <label name> LN in LN_{ij} , if LN is not contained in the label set of GE_{ij} , then add LN to the label set of GE_{ij} .
- 6) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

14.5 <remove statement>

Function

Remove graph element properties and labels.

Format

```
« WG3:BER-036 »

<remove statement> ::=  
  REMOVE <remove item list>

<remove item list> ::=  
  <remove item> [ { <comma> <remove item> }... ]

<remove item> ::=  
  <remove property item> | <remove label item>

<remove property item> ::=  
  <binding variable> <period> <property name>

« WG3:BER-036 »

<remove label item> ::=  
  <binding variable> <is or colon> <label set specification>
```

** Editor's Note (number 96) **

Consider allowing a single optional <where clause>. See Language Opportunity [GQL-169](#).

Syntax Rules

** Editor's Note (number 97) **

BER-019 has established declared type propagation for working record, `working_table`, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-263](#).

- 1) Let *RS* be the <remove statement>.
- 2) Let *RIL* be the <remove item list> immediately contained in *RS*.
« WG3:BER-036 deleted one SR »
« WG3:BER-036 »
- 3) For every <remove item> *RI* immediately contained in *RIL*, the declared type of <binding variable> immediately contained in *RI* shall be a graph element type.

** Editor's Note (number 98) **

A corresponding type check for the dynamically `type case` needs to be added to the General Rules when dynamic types are sufficiently defined. See Possible Problem [GQL-007](#).

General Rules

- 1) Let *TABLE* be the current working table.

- 2) Let n be the number of records of $TABLE$ and let m be the number of <remove item>s immediately contained in RIL .
- 3) For each i -th record R_i , $1 \leq i \leq n$, of $TABLE$ in a new child execution context amended with R_i :
 - a) For each j -th <remove item> RI_j , $1 \leq j \leq m$, immediately contained in RIL determine the graph element $GE_{i,j}$ as follows

Case:

- i) If RI_j immediately contains a <remove property item> RPI_j , let $GE_{i,j}$ be the value of the <binding variable> immediately contained in RPI_j .
- ii) If RI_j immediately contains a <remove label item> RLI_j , let $GE_{i,j}$ be the value of the <binding variable> immediately contained in RLI_j .

« WG3:BER-036 deleted one subrule »

- b) For each j -th <remove item> RI_j , $1 \leq j \leq m$, immediately contained in RS , if RI_j immediately contains a <remove label item> RLI_j , then let the label names $LN_{i,j}$ be the collection of all <label name>s simply contained in RLI_j ; otherwise, let $LN_{i,j}$ be undefined.
- 4) For each i -th record R_i , $1 \leq i \leq n$, of $TABLE$ (using the same order as General Rule 3), perform the following data-modifying operations in a new child execution context amended with R_i for each j -th <remove item> RI_j , $1 \leq j \leq m$, immediately contained in RIL .

Case:

- a) If RI_j immediately contains a <remove property item> RPI_j , then let PN_j be the <property name> immediately contained in RPI_j and if $GE_{i,j}$ is not the null value, then remove the property PN_j of $GE_{i,j}$.
- b) If RI_j immediately contains a <remove label item> and $GE_{i,j}$ is not the null value, then remove the labels $LN_{i,j}$ of $GE_{i,j}$.

- 5) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

14.6 <delete statement>

Function

Delete graph elements.

Format

```
<delete statement> ::=  
  [ DETACH ] DELETE <delete item list> [ <when clause> ]  
  
<delete item list> ::=  
  <delete item> [ { <comma> <delete item> }... ]  
  
<delete item> ::=  
  <value expression>
```

**** Editor's Note (number 99) ****

Consider allowing a single optional <where clause>. See Language Opportunity **GQL-169**.

Syntax Rules

**** Editor's Note (number 100) ****

>BER-019 has established declared type propagation for working record, working table, and result. This Subclause does not do that yet and requires corresponding Syntax Rules.

- 1) Let *DS* be the <delete statement>.
- 2) If *DS* immediately contains DETACH, then let *PREFIX* be DETACH; otherwise, let *PREFIX* be the zero-length character string.
- 3) Let *DIL* be the <delete item list> immediately contained in *DS*.
- 4) If *DS* immediately contains a <when clause>, then let *SC* be the <search condition> immediately contained in *WC*. *DS* is effectively replaced by the <do statement>:

```
DO {  
  FILTER SC  
  PREFIX DELETE DIL  
}
```

General Rules

- 1) Let *TABLE* be the current working table.
- 2) Let *n* be the number of records of *TABLE* and let *m* be the number of <delete item>s immediately contained in *DIL*.
- 3) For each *i*-th record *R_i*, $1 \leq i \leq n$, of *TABLE* in a new child execution context amended with *R_i* and for each *j*-th <delete item> *DI_j*, $1 \leq j \leq m$, immediately contained in *DIL*, let the value *V_{i,j}* be the result of the <value expression> immediately contained in *DI_j*.

- 4) For each i -th record R_i , $1 \leq i \leq n$, of $TABLE$ and for each j -th <delete item> DI_j , $1 \leq j \leq m$, immediately contained in DIL in a new child execution context amended with R_i :

a) If V_{ij} is a reference value to a node N and DETACH is specified, then N and any edges connected to it are deleted.

b) If V_{ij} is a reference value to a node N and DETACH is not specified, then

Case:

i) If N has no edges, then N is deleted.

ii) Otherwise, an exception condition is raised: *dependent object error — edges still exist* (G1001).

c) If V_{ij} is a reference value to an edge E , then E is deleted.

d) If V_{ij} is a path P , then perform all of the following steps completely unless an exception condition is raised:

NOTE 93 — This specifies that P is to be deleted atomically with respect to following operations of the currently active GQL-transaction, i.e., if any of its elements cannot be deleted, none of its constituting elements are.

**** Editor's Note (number 101) ****

It needs to be determined if this is specification of deletion is sufficiently precise, e.g., regarding correctly updating descriptors. See Possible Problem [GQL-165](#).

i) For each reference value E to an edge of P , delete its referent by applying [General Rule 4\)c\)](#) with V_{ij} defined as E .

ii) For each reference value N to a node of P , if DETACH is specified, then delete its referent by applying [General Rule 4\)a\)](#) with V_{ij} defined as N ; otherwise, delete its referent by applying [General Rule 4\)b\)](#) with V_{ij} defined as N .

e) If V_{ij} is the null value, then do nothing.

**** Editor's Note (number 102) ****

Determine if this needs to be spelled out or may be implied by typing rules. See Possible Problem [GQL-007](#).

- 5) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

14.7 <call data-modifying procedure statement>

Function

Define a <call data-modifying procedure statement>.

Format

```
<call data-modifying procedure statement> ::=  
  <call procedure statement>
```

Syntax Rules

None.

General Rules

- 1) Let *CDPS* be the <call data-modifying procedure statement>, let *CPS* be the <call procedure statement> immediately contained in *CDPS*, and let *PC* be the <procedure call> immediately contained in *CPS*.
- 2) Case:
 - a) If *PC* is the <inline procedure call> that immediately contains the <nested procedure specification> that does not immediately contain the <data-modifying procedure specification>, then an exception condition is raised:

**** Editor's Note (number 103) ****
 Exception condition to be determined. See Possible Problem [GQL-158](#).
 - b) If *PC* is the <named procedure call> that immediately contains the <procedure reference> *PR* and the result of *PR* is not a data-modifying procedure, then an exception condition is raised:

**** Editor's Note (number 104) ****
 Exception condition to be determined. See Possible Problem [GQL-158](#).
- 3) The General Rules of *CPS* are applied.

Conformance Rules

None.

15 Query statements

15.1 <composite query statement>

Function

« BER-019 »

Set the result of a <composite query expression> as the current working table.

Format

```
<composite query statement> ::=  
  <composite query expression>
```

Syntax Rules

« BER-019 »

- 1) Let CQS be the <composite query statement> and let CQE be the <composite query expression> immediately contained in CQS .
- 2) The declared type of the incoming working record of CQE is the declared type of the incoming working record of CQS .
- 3) The declared type of the incoming working table of CQE is the declared type of the incoming working table of CQS .
- 4) The declared type of the outgoing working record of CQS is the declared type of the incoming working record of CQS .
- 5) The declared type of the outgoing working table of CQS is the declared type of CQE .
- 6) The declared type of CQS is the declared type of CQE .

General Rules

« BER-019 One rule deleted »

- 1) The General Rules of CQE are applied; let NEW_TABLE be the result of the application of these General Rules.
- 2) Set the current working table to NEW_TABLE .
- 3) Set the current execution outcome to a successful outcome with NEW_TABLE as its result.

Conformance Rules

None.

« WG3:BER-049 deleted one Subclause »

15.2 <composite query expression>

Function

Define a <composite query expression>.

Format

```
<composite query expression> ::=  
    <composite query expression> <query conjunction> <linear query expression>  
    | <linear query expression>  
  
<query conjunction> ::=  
    <set operator>  
    | OTHERWISE  
  
<set operator> ::=  
    UNION [ <set quantifier> ]  
    | EXCEPT [ <set quantifier> ]  
    | INTERSECT [ <set quantifier> ]
```

Syntax Rules

**** Editor's Note (number 105) ****

BER-019 has established declared type propagation for working record, working table, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-266](#).

- 1) If <set operator> is specified and <set quantifier> is not specified, then DISTINCT is implicit.
- 2) Let *CQE* be the <composite query expression>.
- 3) If a <query conjunction> *QC* is immediately contained in *CQE*, then every <query conjunction>s directly contained in *CQE* shall be *QC*.
- 4) If *CQE* directly contains a <focused linear query statement>, then *CQE* shall not directly contain an <ambient linear query statement>.
- 5) If *CQE* directly contains an <ambient linear query statement>, then *CQE* shall not directly contain a <focused linear query statement>.
- 6) If *CQE* directly contains a <primitive result statement> that is a <project statement> or END, then *CQE* shall directly contain at most one <linear query expression>.

General Rules

- 1) Case:
 - a) If <query conjunction> is specified, then:
 - i) Let *ICQE* be the <composite query expression> immediately contained in *CQE* and let *LQE* be the <linear query expression> immediately contained in *CQE*.
 - ii) Let *ICQER* be a new binding table set to the result of *ICQE*.
 - iii) Let *LQER* be a new binding table set to the result of *LQE*.

15.2 <composite query expression>

- iv) Let $FCQER$ be a new binding table.
- v) If <set operator> is specified, then:
 - 1) If $ICQER$ and $LQER$ do not have identical sets of column names, then an exception condition is raised: *syntax error or access rule violation — invalid reference (42002)*.

**** Editor's Note (number 106) ****

This needs to be made more explicit, including how the columns of the two tables are related and what the result of data type combination is, and it should preferably be made into a Syntax Rule.
Possible Problem [GQL-228](#).

- 2) $FCQER$ contains the following records:
 - A) Let R be a record that is a duplicate of some record in $ICQER$ or of some record in $LQER$ or both. Let m be the number of duplicates of R in $ICQER$ and let n be the number of duplicates of R in $LQER$, where $m \geq 0$ (zero) and $n \geq 0$ (zero).
 - B) If DISTINCT is specified or implicit, then
 - Case:
 - I) If UNION is specified, then
 - Case:
 - 1) If $m > 0$ (zero) or $n > 0$ (zero), then $FCQER$ contains exactly one duplicate of R .
 - 2) Otherwise, $FCQER$ contains no duplicate of R .
 - II) If EXCEPT is specified, then
 - Case:
 - 1) If $m > 0$ (zero) and $n = 0$ (zero), then $FCQER$ contains exactly one duplicate of R .
 - 2) Otherwise, $FCQER$ contains no duplicate of R .
 - III) If INTERSECT is specified, then
 - Case:
 - 1) If $m > 0$ (zero) and $n > 0$ (zero), then $FCQER$ contains exactly one duplicate of R .
 - 2) Otherwise, $FCQER$ contains no duplicates of R .
 - C) If ALL is specified, then
 - Case:
 - I) If UNION is specified, then the number of duplicates of R that $FCQER$ contains is $(m + n)$.
 - II) If EXCEPT is specified, then the number of duplicates of R that $FCQER$ contains is the maximum of $(m - n)$ and 0 (zero).
 - III) If INTERSECT is specified, then the number of duplicates of R that $FCQER$ contains is the minimum of m and n .

vi) If OTHERWISE is specified, then

Case:

1) If *ICQER* contains at least one record, then let *FCQER* be *ICQER*.

2) Otherwise, let *FCQER* be *LQER*.

vii) The result of the application of this Subclause is *FCQER*.

b) Otherwise, the result of the application of this Subclause is the result of the <linear query expression>.

Conformance Rules

None.

15.3 <linear query expression>

Function

Define a <linear query expression>.

Format

```
<linear query expression> ::=  
  <linear query statement>
```

Syntax Rules

None.

General Rules

- 1) Let LQE be the <linear query expression> and let LQS be the <linear query statement> immediately contained in LQE .
- 2) Let $TABLE$ be the current working table.
- 3) The General Rules of LQS are applied in a new child execution context $CONTEXT$ with $TABLE$ as its working table; let $RESULT$ be the execution result available in $CONTEXT$ after the application of these General Rules.

NOTE 94 — Re-using the current working table in $CONTEXT$ ensures that all operands of a <composite query expression> are evaluated on it.

- 4) The result of the application of this Subclause is $RESULT$.

Conformance Rules

None.

15.4 <linear query statement>

Function

« BER-019 »

Specify a linear composition of <simple query statement>s that returns a result.

Format

```
<linear query statement> ::=  

  <focused linear query statement>  

  | <ambient linear query statement>  

<focused linear query statement> ::=  

  [ <focused linear query statement part>... ] <focused linear query and primitive result  

  statement part>  

  | <focused primitive result statement>  

  | <focused nested query specification>  

  | <select statement>  

<focused linear query statement part> ::=  

  <use graph clause> <simple linear query statement>  

<focused linear query and primitive result statement part> ::=  

  <use graph clause> <simple linear query statement> <primitive result statement>  

<focused primitive result statement> ::=  

  <use graph clause> <primitive result statement>  

<focused nested query specification> ::=  

  <use graph clause> <nested query specification>  

<ambient linear query statement> ::=  

  [ <simple linear query statement> ] <primitive result statement>  

  | <nested query specification>  

<simple linear query statement> ::=  

  <simple query statement>...
```

Syntax Rules

« BER-019 »

- 1) Let LQS be the <linear query statement>.
- 2) Let $STMSEQ$ be the sequence of <simple query statement>s, the <primitive result statement>, the <nested query specification>, and the <select statement> directly contained in LQS , let m be the number of elements of $STMSEQ$ and let STM_j , $1 \leq j \leq m$, be the j -th element of $STMSEQ$.
- 3) The declared type of the incoming working record of STM_1 is the declared type of the incoming working record of LQS .
- 4) The declared type of the incoming working table of STM_1 is the declared type of the incoming working table of LQS .
- 5) For $2 \leq j \leq m$:

- a) The declared type of the incoming working record of STM_j is the declared type of the outgoing working record of STM_{j-1} .
 - b) The declared type of the incoming working table of STM_j is the declared type of the outgoing working table of STM_{j-1} .
- 6) The declared type of LQS is the declared type of STM_m .

General Rules

None.

Conformance Rules

None.

15.5 Data-reading statements

15.5.1 <match statement>

Function

Expand the current working table with matches from a graph pattern.

Format

```
<match statement> ::=  
[ <statement mode> ] MATCH <graph pattern>
```

Syntax Rules

**** Editor's Note (number 107) ****

BER-019 has established declared type propagation for working record, working table, and result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-267](#).

- 1) Let *MS* be the <match statement>.
- 2) Let *GP* be the <graph pattern> that is immediately contained in *MS*.
- 3) If *MS* immediately contains a <statement mode> *SM*, then

Case:

- a) If *SM* is OPTIONAL, then *MS* is effectively replaced by the <optional statement>:

```
OPTIONAL { MATCH GP RETURN * }
```

- b) Otherwise, *MS* is effectively replaced by the <mandatory statement>:

```
MANDATORY { MATCH GP RETURN * }
```

- 4) For each <element property specification> *EPS* simply contained in *GP*:

- a) Let *EPP* be the <element pattern predicate> that simply contains *EPS*.
- b) Let *EPF* be the <element pattern filler> that simply contains *EPP*.
- c) Let *EVAR* be determined as follows

Case:

- i) If *EPF* simply contains an <element variable declaration>, then *EVAR* is the <identifier> contained in the <element variable declaration> simply contained in *EPF*.
- ii) Otherwise, *EVAR* is an implementation-dependent (UV008) <identifier> distinct from every element variable, subpath variable, and path variable contained in *GP*.

- d) Let *PECL* be the <property key value pair list> simply contained in *EPS*.
- e) Let *NOPEC* be the number of <property key value pair>s simply contained in *PECL*.
- f) Let *PEC₁*, ..., *PEC_{NOPEC}* be the <property key value pair>s simply contained in *PECL*.
- g) For every *i*, $1 \leq i \leq NOPEC$:

15.5 Data-reading statements

- i) Let $PROP_i$ be the <property name> simply contained in PEC_i .
 - ii) Let VAL_i be the <value expression> simply contained in PEC_i .
 - iii) Let $RPEC_i$ be a <comparison predicate> formed as:

$$\text{EVAR}.\text{PROP}_i = \text{VAL}_i$$
- h) Let $EPSC$ be a <boolean value expression> formed through the concatenation of <boolean factor>s: $RPEC_1$ AND ... AND ... $RPEC_{NOPEC}$.
- i) EPP is effectively replaced by:

`WHERE EPSC`

General Rules

- 1) Let $TABLE$ be the current working table.
- 2) Let NEW_TABLE be a new empty binding table.
- 3) For each record R of $TABLE$ in a new child execution context amended with R :
 - a) Let M be the result of GP .
 - b) Append the Cartesian product of R and M to NEW_TABLE .
- 4) Set the current working table to NEW_TABLE .
- 5) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

15.5.2 <call query statement>

** Editor's Note (number 108) **

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:BNE-023], needs further discussion to establish consensus. See Possible Problem [GQL-107](#).

Function

Define a <call query statement>.

Format

```
<call query statement> ::=  
  <call procedure statement>
```

Syntax Rules

None.

General Rules

- 1) Let *CQS* be the <call query statement>, let *CPS* be the <call procedure statement> immediately contained in *CQS*, and let *PC* be the <procedure call> immediately contained in *CPS*.
- 2) Case:
 - a) If *PC* is the <inline procedure call> that immediately contains the <nested procedure specification> that does not immediately contain the <query specification>, then an exception condition is raised:

** Editor's Note (number 109) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

- b) If *PC* is the <named procedure call> that immediately contains the <procedure reference> *PR* and the result of *PR* is not a query, then an exception condition is raised:

** Editor's Note (number 110) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

- 3) The General Rules of *CPS* are applied.

Conformance Rules

None.

15.6 Data-transforming statements

15.6.1 <mandatory statement>

** Editor's Note (number 111) **

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:BNE-023], needs further discussion to establish consensus. See Possible Problem [GQL-108](#).

Function

« BER-019 »

Raise an exception condition if the specified <procedure call> sets the current working table to an empty binding table.

Format

```
<mandatory statement> ::=  
    MANDATORY <procedure call>
```

** Editor's Note (number 112) **

Consider allowing a single optional <where clause>. See Language Opportunity [GQL-169](#).

Syntax Rules

- 1) Let *MS* be the <mandatory statement>.
- 2) Let *PC* be the <procedure call> immediately contained in *MS*.
 « BER-019 »
- 3) The declared type of the incoming working record of *PC* is the declared type of the incoming working record of the *MS*.
- 4) The declared type of the incoming working table of *PC* is the declared type of the incoming working table of the *MS*.
- 5) The declared type of *PC* shall be a binding table type.
 « BER-019 »
- 6) The declared type of the outgoing working record of *MS* is the declared type of the incoming working record of the *MS*.
- 7) The declared type of the outgoing working table of *MS* is the declared type of *PC*.
- 8) *MS* has no declared type.

General Rules

- 1) Let *TABLE* be the current working table.
- 2) Let *NEW_TABLE* be a new empty binding table.
- 3) For each record *R* of *TABLE*, in a new child execution context amended with *R*:

- a) Let $RESULT$ be the result of PC .
 - b) Case:
 - i) If $RESULT$ is an empty binding table, then an exception condition is raised: *data exception — empty binding table returned (22G0D)*.
 - ii) Otherwise, the Cartesian product of R and $RESULT$ is appended to NEW_TABLE .
- 4) Set the current working table to NEW_TABLE .
 - 5) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

15.6.2 <optional statement>

** Editor's Note (number 113) **

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:BNE-023], needs further discussion to establish consensus. See Possible Problem [GQL-109](#).

Function

« BER-019 »

Replace each record of the current working table with a set of new records with additional fields resulting from the specified <procedure call> unless that set is empty in which case the record is extended with fields containing the null value instead.

Format

```
<optional statement> ::=  
    OPTIONAL <procedure call>
```

** Editor's Note (number 114) **

Consider allowing a single optional <where clause>. See Language Opportunity [GQL-169](#).

Syntax Rules

- 1) Let *OS* be the <optional statement>.
- 2) Let *PC* be the <procedure call> immediately contained in *OS*.
 « BER-019 »
- 3) The declared type of the incoming working record of *PC* is the declared type of the incoming working record of the *OS*.
- 4) The declared type of the incoming working table of *PC* is the declared type of the incoming working table of the *OS*.
- 5) Let *DTPC* be the declared type of *PC*.
- 6) *DTPC* shall be a binding table type.
 « BER-019 »
- 7) The declared type of the outgoing working record of *OS* is the declared type of the incoming working record of the *OS*.
- 8) The declared type of the outgoing working table of *OS* is the declared type of *PC*.
- 9) *OS* has no declared type.

General Rules

- 1) Let *TABLE* be the current working table.
- 2) Let *NEW_TABLE* be a new empty binding table.
- 3) Let *FTSET* be the set of field types of the record type of *DTPC*.

- 4) For each record R of $TABLE$ in a new child execution context amended with R :
 - a) Let $RESULT$ be the result of PC .
 - b) Case:

« BER-019 »

- i) If $RESULT$ is an empty result table, then
 - 1) Let NR be a record that extends R as follows:
 - A) For each field type FT in $FTSET$ there is one additional field F in NR .
 - B) The field name of F is the field name of FT and the field value of F is the null value.
 - 2) NR is appended to NEW_TABLE .
 - ii) Otherwise, the Cartesian product of R and $RESULT$ is appended to NEW_TABLE .
- 5) Set the current working table to NEW_TABLE .
- 6) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

15.6.3 <filter statement>

Function

Select a subset of the records of the current working table.

Format

```
<filter statement> ::=  
  FILTER { <where clause> | <search condition> }
```

Syntax Rules

- 1) Let FS be the <filter statement>.
- 2) If FS immediately contains the <search condition> SC , then it effectively is replaced by the <filter statement>:

FILTER WHERE SC

« BER-019 »

- 3) Let WC be the <where clause> that is immediately contained in FS .
- 4) Let $DTIWR$ be the declared type of the incoming working record of the FS .
- 5) Let $DTIWT$ be the declared type of the incoming working table of the FS .
- 6) The declared type of the incoming working record of WC is $DTIWR$.
- 7) The declared type of the incoming working table of WC is $DTIWT$.
- 8) The declared type of the outgoing working record of FS is $DTIWR$.
- 9) The declared type of the outgoing working table of FS is $DTIWT$.

General Rules

« BER-019 One rule deleted »

- 1) Set the current working table to the result of WC .

Conformance Rules

None.

15.6.4 <let statement>

**** Editor's Note (number 115) ****

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:BNE-023] as WITH, needs further discussion to establish consensus. See Possible Problem [GQL-111](#).

Function

Add columns to the current working table.

Format

```
<let statement> ::=  

    LET <compact variable definition list>  

    | <statement mode> LET <compact variable definition list> <where clause>  

<compact variable definition list> ::=  

    <compact variable definition> [ { <comma> <compact variable definition> }... ]
```

Syntax Rules

- 1) Let LS be the <let statement> and let $CVDL$ be the <compact variable definition list> immediately contained in LS .
- 2) Let $CVDSEQ$ be the sequence of <compact variable definition>s immediately contained in $CVDL$ and let n be the number of elements of $CVDSEQ$.
- 3) Let CVD_i , $1 \leq i \leq n$, be the i -th element of $CVDSEQ$.
- 4) Case:
 - a) If LS immediately contains the <statement mode> OPTIONAL, then let WC be the <where clause> immediately contained in LS , let SC be the <search condition> immediately contained in WC . LS is effectively replaced by:

```
OPTIONAL {  

    FILTER SC  

    LET CVDL  

    RETURN *  

}
```

- b) If LS immediately contains the <statement mode> MANDATORY, then let WC be the <where clause> immediately contained in LS , let SC be the <search condition> immediately contained in WC . LS is effectively replaced by:

```
MANDATORY {  

    FILTER SC  

    LET CVDL  

    RETURN *  

}
```

- c) Otherwise, let the new <binding variable definition block> $DBLK$ be the <newline>-separated list of all CVD_i , $1 \leq i \leq n$. LS is effectively replaced by:

```
CALL {  

    DBLK
```

```
    RETURN *  
}
```

General Rules

None.

Conformance Rules

None.

15.6.5 <aggregate statement>

**** Editor's Note (number 116) ****

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:BNE-023] as WITH, needs further discussion to establish consensus. See Possible Problem [GQL-112](#).

Function

Aggregate over the current working table.

Format

```
<aggregate statement> ::=  

    AGGREGATE <compact value variable definition> [ { <comma> <compact value variable  

    definition> }... ] <where clause>
```

**** Editor's Note (number 117) ****

Consider extending aggregate with GROUP BY. See Language Opportunity [GQL-199](#).

Syntax Rules

**** Editor's Note (number 118) ****

BER-019 has established declared type propagation for working record, working table, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-268](#).

- 1) Let *AS* be the <aggregate statement>.
- 2) Let *CVVSEQ* be the sequence of every <compact value variable definition> immediately contained in *AS* and let *NCVVSEQ* be the number of elements of *CVVSEQ*.
- 3) Let $CVVN_i$, $1 \leq i \leq NCVVSEQ$, be the <value variable> immediately contained in the *i*-the element of *CVVSEQ* and let *CVVE_i* be the <value expression> immediately contained in the *i*-the element of *CVVSEQ*.
- 4) $CVVN_i$, $1 \leq i \leq NCVVSEQ$, shall not be the name of a binding variable that is in scope.
- 5) $CVVE_i$, $1 \leq i \leq NCVVSEQ$, shall be an <aggregate function>.
- 6) $CVVN_i$, $1 \leq i \leq NCVVSEQ$, is a new fixed variable in scope after *AS* whose declared type is the declared type of *CVVE_i*.

General Rules

- 1) Let *TABLE* be the current working table.
- 2) Let *FILTERED_TABLE* be determined as follows

Case:

 - a) If *AS* immediately contains the <where clause> *WC*, then let *FILTERED_TABLE* be the result of *WC*.

15.6 Data-transforming statements

- b) Otherwise, let *FILTERED_TABLE* be *TABLE*.
- 3) Set the current working table to *FILTERED_TABLE*.
- 4) Let *R* be a new record with fields F_i , $1 \leq i \leq NCVVSEQ$, such that the field name of F_i is the name of $CVVN_i$ and the field value of F_i is the result of $CVVE_i$.
- 5) Let *RESULT_TABLE* be the Cartesian Product between *TABLE* and *R*.
- 6) Set the current working table to *RESULT_TABLE*.
- 7) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

15.6.6 <for statement>

**** Editor's Note (number 119) ****

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:BNE-023] as UNWIND, needs further discussion to establish consensus. See Possible Problem [GQL-113](#).

Function

Provide iteration over (unnesting of) collections by expanding the current working table.

**** Editor's Note (number 120) ****

FOR was included by the editors as an additional variation of UNNEST that directly operates on the current working table instead of being a Subclause of SELECT. A neutral keyword was chosen, which is not present in any input language of GOL, with the intention of providing syntax that can serve as a starting point for further discussion on this topic. See Possible Problem [GQL-113](#).

Format

```
<for statement> ::=  
  [ <statement mode> ] FOR <for item list> [ <for ordinality or index> ] [ <where clause> ]  
  
<for item list> ::=  
  <for item> [ { AND <for item> }... ]  
  « Consequence of WG3:BER-094R1 »  
  
<for item> ::=  
  <for item alias> <list value expression>  
  
<for item alias> ::=  
  <identifier> IN  
  
<for ordinality or index> ::=  
  WITH { ORDINALITY | INDEX } [ <identifier> ]
```

**** Editor's Note (number 121) ****

<for ordinality or index> should be moved into <for item>. See Possible Problem [GQL-113](#).

Syntax Rules

**** Editor's Note (number 122) ****

BER-019 has established declared type propagation for working record, working table, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-269](#).

- 1) Let FS be the <for statement>.
- 2) Let FIL be the <for item list> that is immediately contained in FS and let $NFIL$ be the number of elements of FIL .
- 3) If <for ordinality or index> if specified, then let FOI be <for ordinality or index>; otherwise, let FOI be the zero-length character string.

15.6 Data-transforming statements

- 4) If <where clause> if specified, then let *WC* be <where clause>; otherwise, let *WC* be the zero-length character string.
- 5) Case:
 - a) If *FS* immediately contains a <statement mode> *SM*, then

Case:

 - i) If *SM* is OPTIONAL, then *FS* is effectively replaced by the <optional statement>:

```
OPTIONAL { FOR FIL FOI WC RETURN * }
```

 - ii) If *SM* is MANDATORY, then *FS* is effectively replaced by the <mandatory statement>:

```
MANDATORY { FOR FIL FOI WC RETURN * }
```
 - b) Otherwise:
 - i) If *WC* immediately contains the <search condition> *SC*, then *FS* is effectively replaced by:


```
FILTER WHERE SC
FOR FIL FOI
```
 - ii) If *FOI* immediately specifies WITH ORDINALITY but does not immediately contain an <identifier>, then *FS* is effectively replaced by the <for statement>:


```
FOR FIL WITH ORDINALITY ordinality
```
 - iii) If *FOI* immediately specifies WITH INDEX but does not immediately contain an <identifier>, then *FS* is effectively replaced by the <for statement>:


```
FOR FIL WITH INDEX index
```

General Rules

- 1) Let *NEW_TABLE* be a new binding table.
 « Consequence of WG3:BER-094R1 »
- 2) Let $FI_i, 1 \leq i \leq NFIL$, be the *i*-th <for item> of *FIL*, let *FCVE_i* be the <list value expression> immediately contained in *FI_i*, and let *FIA_i* be the <for item alias> immediately contained in *FI_i*.
- 3) For each record *R* of the current working table in a new child execution context:
 - a) Set the current working record to *R*.
 - b) Let *FCVER_i*, $1 \leq i \leq NFIL$, be the result of *FCVE_i*.
 - c) Let the list $LV_i, 1 \leq i \leq NFIL$, be determined as follows.

Case:

 - i) If *FCVER_i* is a list, then *LV_i* is *FIVER_i*.
 - ii) If *FCVER_i* is the null value, then *LV_i* is the empty list.
 - iii) Otherwise, an exception condition is raised: *data exception — invalid value type (22G03)*.

- d) Let $LVMAXLEN$ be the largest length of any list LV_i , $1 \leq i \leq NFIL$.
- e) For every j , $1 \leq j \leq LVMAXLEN$:
 - i) For each i , $1 \leq i \leq NVIL$, if j is less than or equal to the length of LV_i , then let $LVE_{i,j}$ be the j -th element of LV_i . Otherwise let $LVE_{i,j}$ be the null value.
 - ii) Let LR_j be the record with the fields $LRF_{i,j}$, $1 \leq i \leq NFIL$, such that the field name of each field $LRF_{i,j}$ is the <identifier> immediately contained in FIA_i and the field value of each field $LRF_{i,j}$ is $LVE_{i,j}$.
 - iii) If the <for ordinality or index> is specified, then
 - Case:
 - 1) If FOI immediately contains WITH ORDINALITY, then let LRO_j be the record obtained by adding a field to LR_j whose field name is the <identifier> that is immediately contained in FOI and whose field value is j represented as a value of an implementation-defined (ID057) exact numeric type with scale 0 (zero).
 - 2) If FOI immediately contains WITH INDEX, then let LRO_j be the record obtained by adding a field to LR_j whose field name is the <identifier> that is immediately contained in FOI and whose field value is $j-1$ represented as a value of an implementation-defined (ID058) exact numeric type with scale 0 (zero).
 - 3) Otherwise, let LRO_j be LR_j .
 - iv) Append LRO_j to the current working table.
 - f) Append the Cartesian product of R and the current working table to NEW_TABLE .
- 4) Set the current working table to NEW_TABLE .

Conformance Rules

None.

15.6.7 <order by and page statement>

Function

Order the records of the current working table according to a provided sort specification and optionally retain only a specified number of records.

Format

```
<order by and page statement> ::=  
  <order by clause> [ <offset clause> ] [ <limit clause> ]  
  | <offset clause> [ <limit clause> ]  
  | <limit clause>
```

** Editor's Note (number 123) **

Additional support for PARTITION BY, WITH TIES, WITH INDEX, and WITH ORDINALITY should be considered. See Language Opportunity [GOL-163](#).

Syntax Rules

« BER-019 »

- 1) Let *OPS* be the <order by and page statement>.
- 2) Let *WC* be the <where clause> that is immediately contained in *OPS*.
- 3) Let *DTIWR* be the declared type of the incoming working record of the *OPS*.
- 4) Let *DTIWT* be the declared type of the incoming working table of the *OPS*.
- 5) If *OPS* immediately contains the <order by clause> *OBC*, then
 - a) The declared type of the incoming working record of *OBC* is *DTIWR*.
 - b) The declared type of the incoming working table of *OBC* is *DTIWT*.
- 6) If *OPS* immediately contains the <offset clause> *OC*, then
 - a) The declared type of the incoming working record of *OC* is *DTIWR*.
 - b) The declared type of the incoming working table of *OC* is *DTIWT*.
- 7) If *OPS* immediately contains the <limit clause> *LC*, then
 - a) The declared type of the incoming working record of *LC* is *DTIWR*.
 - b) The declared type of the incoming working table of *LC* is *DTIWT*.
- 8) The declared type of the outgoing working record of *OPS* is *DTIWR*.
- 9) The declared type of the outgoing working table of *OPS* is *DTIWT*.
- 10) *OPS* has no declared type.

General Rules

« BER-019 One rule deleted »

- 1) Let *TABLE* be the current working table.
- 2) Let *ORDERED* be the binding table determined as follows

Case:

- a) If *OPS* immediately contains the <order by clause> *OBC*, let *ORDERED* be the result of *OBC*.
- b) Otherwise, let *ORDERED* be *TABLE*.

- 3) Let *OFFSETED* be the binding table determined as follows

Case:

- a) If *OPS* immediately contains the <offset clause> *OC*, let *OFFSETED* be the result of *OC*.
- b) Otherwise, let *OFFSETED* be *ORDERED*.

- 4) Let *LIMITED* be the binding table determined as follows

Case:

« BER-019 »

- a) If *OPS* immediately contains the <limit clause> *LC*, then let *LIMITED* be the result of *LC*.
- b) Otherwise, let *LIMITED* be *OFFSETED*.

- 5) Set the current working table to *LIMITED*.

- 6) Set the current execution outcome to a successful outcome with an omitted result.

Conformance Rules

None.

15.7 Result projection statements

15.7.1 <primitive result statement>

** Editor's Note (number 124) **

WG3:W05-020 suggests that discussion of this Subclause is needed to establish consensus. See Possible Problem [GQL-072](#).

Function

« BER-019 »

Define what to include in a query result.

Format

```
<primitive result statement> ::=  
    <return statement> [ <order by and page statement> ]  
    | <project statement>  
    | END
```

Syntax Rules

« BER-019 »

- 1) Let *PRS* be the <primitive result statement>.
- 2) Let *DTIWR* be the declared type of the incoming working record of the *PRS*.
- 3) Let *DTIWT* be the declared type of the incoming working table of the *PRS*.
- 4) If the <return statement> *RS* is specified:
 - a) The declared type of the incoming working record of *RS* is *DTIWR*.
 - b) The declared type of the incoming working table of *RS* is *DTIWT*.
 - c) Case:
 - i) If the <order by and page statement> *OPS* is specified, then
 - 1) The declared type of the incoming working record of *OPS* is *DTIWR*.
 - 2) The declared type of the incoming working table of *OPS* is the declared type of *RS*.
 - 3) The declared type of the result is the declared type of the outgoing working table of *OPS*.
 - ii) Otherwise, the declared type of *PRS* is the declared type of *RS*.
 - 5) If the <project statement> *PS* is specified:
 - a) The declared type of the incoming working record of *PS* is *DTIWR*.
 - b) The declared type of the incoming working table of *PS* is *DTIWT*.
 - c) The declared type of *PRS* is the declared type of *PS*.

- | 6) If END is specified, *PRS* has no declared type.

General Rules

- 1) If END is specified, then the current execution outcome is set to a successful outcome with an omitted result.

Conformance Rules

None.

15.7.2 <return statement>

** Editor's Note (number 125) **

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:JCJ-010r1] and [WG3:BNE-023], needs further discussion to establish consensus. See Possible Problem [GQL-116](#).

Function

Define a <return statement> for determining a result binding table.

Format

```

<return statement> ::=  
  RETURN <return statement body>  
  
<return statement body> ::=  
  [ <set quantifier> ] { <asterisk> | <return item list> }  
  [ <group by clause> ]  
  
<return item list> ::=  
  <return item> [ { <comma> <return item> }... ]  
  
<return item> ::=  
  <value expression> [ <return item alias> ]  
  
<return item alias> ::=  
  AS <identifier>
  
```

** Editor's Note (number 126) **

Consider allowing a single optional <where clause>. See Language Opportunity [GQL-169](#).

Syntax Rules

** Editor's Note (number 127) **

Aggregation functionality should be improved for the needs of GQL. See Language Opportunity [GQL-017](#).

** Editor's Note (number 128) **

BER-019 has established declared type propagation for working record, working table, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-270](#).

- 1) Let *RS* be the <return statement>.
- 2) Let *RSB* be the <return statement body> immediately contained in *RS*.
- 3) If a <set quantifier> is not immediately contained in *RSB*, then ALL is the implicit <set quantifier> of *RSB*.
- 4) Let *SQ* be the explicit or implicit <set quantifier> of *RSB*.
- 5) If *RSB* immediately contains a <group by clause>, then let *GBC* be that <group by clause>; otherwise, let *GBC* be the zero-length character string.
- 6) If *RSB* immediately contains an <asterisk>, then:

- a) RSB shall not immediately contain a <group by clause>.
- b) Let $BVSEQ$ be the permutation of all binding variables in scope for RS , ordered in standard sort order, let $NBVSEQ$ be the number of such binding variables, and let BV_i , $1 \leq i \leq NBVSEQ$, be the i -th such binding variable in $BVSEQ$.
- c) For i , $1 \leq i \leq NBVSEQ$, let the new <return item list> $NRIL$ be a comma-separated list of <return item>s:

BV_i AS BV_i

- d) RS is effectively replaced by the <return statement>:

RETURN SQ $NRIL$ GBC

- 7) Let RIL be the <return item list> immediately contained in RSB , let $NRIL$ be the number of elements of RIL , and let RI_i , $1 \leq i \leq NRIL$, be the i -th element of RIL .
- 8) For any <return item> RI , let the *expression* of RI be the <value expression> immediately contained in RI and if RI immediately contains a <return item alias> RIA , then let the *alias name* of RI be the <identifier> that is immediately contained in RIA .
- 9) For each <return item> RI_i , $1 \leq i \leq NRIL$, from RIL .

Case:

- a) If the expression of RI_i immediately contains a <binding variable> $RIBV_i$ but RI_i does not have an alias name, then RI_i is effectively replaced by the <return item>:

$RIBV_i$ AS $RIBV_i$

- b) Otherwise, RI_i shall have an alias name.

- 10) Let the set of preserved column names $PCNSET$ be the set of all binding variables in scope for RS that correspond to a column name of the current working table but that are different from the alias name of any <return item> in RIL and let $NPCN$ be the number of elements of $PCNSET$.
- 11) Case:

- a) If GBC is the zero-length character string, then:
 - i) Let $GRISET$ be the set of *grouping* <return item>s contained in RIL whose expression contains no <aggregate function> and let $NGRI$ be the number of such <return item>s in $GRISET$.
 - ii) Let $ARISET$ be the set of *aggregating* <return item>s contained in RIL whose expression contains an <aggregate function> and let $NARI$ be the number of such <return item>s in $ARISET$.
 - iii) If $NARI$ is greater than 0 (zero), then:
 - 1) For each j between 1 (one) and $NGRI$, let GRI_j be an enumeration of the <return item>s of $GRISET$ and let $GRIAI_j$ be an enumeration of the alias names of GRI_j .
 - 2) Let GEL be the comma-separated list of <grouping element>s:

$GRIAI_1, \dots, GRIAI_{NGRI}$

- 3) The following <group by clause> is implicit:

Case:

- A) If $NGRI$ is greater than 0 (zero), then:

GROUP BY *GEL*

- B) Otherwise:

GROUP BY ()

- b) Otherwise:

- i) Let $GRISET$ be the set of grouping <return item>s contained in RIL whose alias name is simply contained in $FGBC$ and let $NGRI$ be the number of such <return item>s in $GRISET$.
- ii) Let $ARISET$ be the set of aggregating <return item>s contained in RIL whose alias name is not simply contained in $FGBC$ and let $NARI$ be the number of such <return item>s in $ARISET$.

- 12) Let $XRISSET$ be the set of all <return item>s contained in RIL that are not contained in $GRISET$ and that are not contained in $ARISET$ and let $NXRI$ be the number of such <return item>s in $XRISSET$.
- 13) If a <binding variable> contained in the expression RIE of some <return item> RI of RIL is not either a binding variable in the scope for RS or reference a fixed variable that is defined in RIE , then an exception condition is raised: *syntax error or access rule violation — invalid reference (42002)*.

** Editor's Note (number 129) **

Further work on scoping rules is required. See Possible Problem [GQL-172](#).

General Rules

- 1) Let $TABLE$ be the current working table, let n be the number of records of $TABLE$, and let R_i , 1 (one) $\leq i \leq n$, be the i -th record of $TABLE$ in an order determined by iteration over $TABLE$.
- 2) Let $RETURN_TABLE$ be a new empty binding table.
- 3) Let $FGBC$ be the implicit or explicit *group by clause* that is immediately contained in RS .
- 4) Case:
 - a) If $FGBC$ is the zero-length character string, then for each record R of $TABLE$ in a new child execution context amended with R :
 - i) Let S be a new record with fields F_i such that the field name of F_i , 1 (one) $\leq i \leq n$, is the alias name of RI_i and the field value of F_i is the result of the expression of RI_i for each <return item> RI_i from RIL .
 - ii) If SQ is ALL, then let SP be a new record comprising all fields of S and additional fields F_i , 1 (one) $\leq i \leq NPCN$ such that the field name of F_i is PCN_i and the field value of F_i is the field value of the field with field name PCN_i in R for each preserved column name PCN_i from $PCNSET$; otherwise, let SP be S .
 - iii) Add SP to $RETURN_TABLE$.
 - b) Otherwise:

- i) Let the *grouping record* GR_i , $1 \leq i \leq n$, of a record R_i of $TABLE$ be a new record with fields F_k , $1 \leq k \leq NGRI$, such that the field name of F_k is the alias name of GRI_k and the field value of F_k is the result of the expression of GRI_k in a new child execution context amended with R_i .
- ii) Let GR_TABLE be a new empty binding table of all grouping records GR_i , $1 \leq i \leq n$, of all records R_i of $TABLE$.
- iii) Let $GROUP_BY$ be the result of GBC in a new child execution context with GR_TABLE as its working table.
- iv) Let $KEYS$ be a permutation of $GROUP_BY$ in an implementation-dependent (US003) order and let $NKEYS$ be the number of elements of $KEYS$.
- v) For each i -th element of $KEYS$ K_i , $1 \leq i \leq NKEYS$:
 - 1) Let $PART_i$ be a new binding table comprising only the records R_j , $1 \leq j \leq n$, from $TABLE$, for which the grouping record GR_j for R_j is equivalent to K_i .

**** Editor's Note (number 130) ****

Equivalence (comparison that treats nulls as identical, e.g., when using DISTINCT) to be specified.
See Possible Problem [GQL-178](#).

- 2) Let $NPART_i$ be the number of records of $PART_i$.
 - 3) For each record $P_{i,j}$, $1 \leq j \leq NPART_i$ in a new child execution context:
 - A) Set the current working record to $P_{i,j}$.
 - B) Set the current working table to $PART_i$.

NOTE 95 — This is used to determine the result of <aggregate function>s.
 - C) Let $NR_{i,j}$ be a new record comprising:
 - I) All fields of K_i .
 - II) Additional fields AF_k , $1 \leq k \leq NARI$, such that the field name of AF_k is the alias name of ARI_k and the field value of AF_k is the result of the expression of ARI_k .
 - III) Additional fields XF_k , $1 \leq k \leq NXRI$, such that the field name of XF_k is the alias name of XRI_k and the field value of XF_k is the result of the expression of XRI_k .
 - D) Append $NR_{i,j}$ to $RETURN_TABLE$.
- 5) If SQ is DISTINCT, then set the current working table to a duplicate-free binding table obtained from the set of all records of $RETURN_TABLE$; otherwise, set the current working table to $RETURN_TABLE$.
 - 6) If SQ is ALL and $FGBC$ is the zero-length character string, then let $SANITIZED_TABLE$ be the binding table obtained from the current working table by discarding all columns whose column name is in $PCNSET$ and set the current working table to $SANITIZED_TABLE$.

15.7 Result projection statements

- 7) Let $FINAL_TABLE$ be a new binding table obtained from the current working table by determining the preferred column sequence to be the sequence of alias names of all $\langle return\ item\rangle$ s RI_i , $1 \leq i \leq NRIL$, from RIL in the order of their occurrence in RIL .
- 8) Set the current working table to $FINAL_TABLE$.
- 9) Set the current execution outcome to a successful outcome with $FINAL_TABLE$ as its result.

Conformance Rules

None.

15.7.3 <select statement>

**** Editor's Note (number 131) ****

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:JCJ-010r1], needs further discussion to establish consensus. Furthermore the Syntax and General Rules are probably incomplete and in any case need to be reviewed. See Possible Problem [GQL-117](#).

Function

Provide an SQL-style query over graph data to produce a tabular query result set.

Format

**** Editor's Note (number 132) ****

The <select statement> provides additional syntax for querying graphs that is closer to the syntax of SQL. Further refinement of the specified Format is required. It is suggested that <select statement> is best implemented via syntax transformation to <return statement>. Alternatively, it may be possible to embed SQL's syntax directly and convert the resulting SQL table to a binding table. See Possible Problem [GQL-187](#).

```

<select statement> ::=

  SELECT [ <set quantifier> ] <select item list>
  [ <select statement body> ]
  [ <where clause> ]
  [ <group by clause> ]
  [ <having clause> ]
  [ <order by clause> ]
  [ <offset clause> ] [ <limit clause> ]

<select item list> ::=
  <select item> [ { <comma> <select item> }... ] 

<select item> ::=
  <value expression> [ <select item alias> ] 

<select item alias> ::=
  AS <identifier>

<having clause> ::=
  HAVING <search condition>

<select statement body> ::=
  FROM <select graph match list>
  | <select query specification>

<select graph match list> ::=
  <select graph match> [ { <comma> <select graph match> }... ] 

<select graph match> ::=
  <graph expression> <match statement>

<select query specification> ::=
  FROM <nested query specification>
  | FROM <graph expression> <nested query specification>

```

**** Editor's Note (number 133) ****

Aggregation functionality should be improved for the needs of GQL. See Language Opportunity [GQL-017](#).

Syntax Rules

**** Editor's Note (number 134) ****

A Syntax Rule is needed to prevent ambiguity between the <where clause> immediately contained in the <select statement> and a <graph pattern where clause> contained in the <graph pattern> in the last <match statement> in <select graph match list>. The Syntax Rule should at least prohibit a <graph pattern where clause> in the <graph pattern> in the last <match statement> in <select graph match list>. Note that a <parenthesized path pattern where clause> does not cause ambiguity with <where clause> immediately contained in the <select statement>.

None.

General Rules

None.

Conformance Rules

None.

15.7.4 <project statement>

** Editor's Note (number 135) **

WG3:W05-020 suggests that discussion of this Subclause is needed to establish consensus. See Possible Problem [GQL-073](#).

Function

Define a <project statement> for projecting a result.

Format

```
<project statement> ::=  
    PROJECT <value expression>
```

** Editor's Note (number 136) **

Consider allowing a single optional <where clause>. See Language Opportunity [GQL-169](#).

Syntax Rules

- 1) Let *VE* be the specified <value expression>.
- 2) If *VE* contains a <binding variable> *BV* without an intervening instance of <procedure body>, then one of the following conditions shall be true:
 - a) *BV* identifies an iterated variable and *BV* is either contained in a <dependent value expression> that is contained in *VE* or *BV* is contained in a <value expression> that is immediately contained in a <general set function> that is contained in *VE*.
 - b) *BV* identifies a fixed variable.

General Rules

- 1) Let *ROVE* be the result of *VE*.
WG3:BER-081 deleted one GR and Editor's Note
- 2) Set the current execution outcome to a successful outcome with *ROVE* as its result.

Conformance Rules

None.

16 Common elements

16.1 <use graph clause>

Function

Declare a working graph and its scope.

Format

```
<use graph clause> ::=  
  USE <graph expression>
```

Syntax Rules

- 1) Let *UGC* be the <use graph clause>.
 «WG3:BER-088R1»
- 2) Let *G* be the graph identified by the <graph expression> immediately contained in *UGC*.
- 3) Let *PART* be the instance that immediately contains *UGC*.
- 4) The scope clause of *UGC* is

Case:

- a) If *PART* is simply contained in a <focused linear query statement> *FLQS*, then *FLQS*.
- b) If *PART* is simply contained in a <focused linear data-modifying statement> *FLDMS*, then *FLDMS*.

- 5) The scope of *G* comprises

Case:

- a) If *PART* is a <focused linear query statement part>, then the <simple linear query statement> immediately contained in *PART*.
- b) If *PART* is a <focused linear query and primitive result statement part>, then the <simple linear query statement> and the <primitive result statement> immediately contained in *PART*.
- c) If *PART* is a <focused primitive result statement>, then the <primitive result statement> immediately contained in *PART*.
- d) If *PART* is a <focused nested query specification>, then the <nested query specification> immediately contained in *PART*.
- e) If *PART* is a <focused linear data-modifying statement body>, then the <simple linear query statement>, the <simple data-modifying statement>, the <simple linear data-accessing statement>, and the <primitive result statement> immediately contained in *PART*.
- f) If *PART* is a <focused nested data-modifying procedure specification>, then the <nested data-modifying procedure specification> immediately contained in *PART*.

- 6) *UGC* declares G as a working graph.

General Rules

None.

Conformance Rules

None.

16.2 <at schema clause>

Function

Declare a working schema and its scope.

Format

```
<at schema clause> ::=  
    AT <schema reference>
```

Syntax Rules

- 1) Let *ASC* be the <at schema clause>.«WG3:BER-088R1»
- 2) Let *S* be the schema identified by the <schema reference> that is immediately contained in *ASC*.
- 3) Let *PB* be the <procedure body> immediately containing *ASC*.
- 4) The scope clause of *ASC* is *PB*.
- 5) The scope of *S* comprises *PB*.
- 6) *ASC* declares *S* as a working schema.

General Rules

None.

Conformance Rules

None.

16.3 Named elements

** Editor's Note (number 137) **

WG3:W05-020 suggests that discussion of this Subclause is needed to establish consensus. See Possible Problem [GQL-075](#).

Function

Specify named elements.

Format

```
<binding variable> ::=  
  <binding variable name>  
  
<parameter> ::=  
  <parameter name>
```

Syntax Rules

« BER-019 »

- 1) If the <binding variable> *BV* is specified, then:
 - a) Let *BVN* be the <binding variable name> immediately contained in *BV*.
 - b) Case:
 - i) If the declared type of the incoming working record has a field type *FT* whose field name is *BVN*, then the declared type of *BV* is the field value type of *FT*.
 - ii) If the current working schema contains a visible catalog object *CO* whose name is *BVN*, then the declared type of *BV* is the declared type of *CO*.
 - iii) Otherwise, an exception condition is raised: *syntax error or access rule violation — invalid reference (42002)*.
- 2) If the <parameter> *P* is specified, then:
 - a) Let *PN* be the <parameter name> immediately contained in *P*.
 - b) Case:
 - i) If there is a request parameter *RP* with parameter name *PN* in the current request context, then the declared type of *P* is the declared type of *RP*.
 - ii) If there is a session parameter *SP* with parameter name *PN* in the current session context, then the declared type of *P* is the declared type of *SP*.
 - iii) Otherwise, an exception condition is raised: *syntax error or access rule violation — invalid reference (42002)*.

General Rules

- 1) If the <binding variable> *BV* is specified, then
 - Case:

- a) If the current working record has a field F whose field name is BV , then the result of BV is the field value of F .

« BER-019 »

- b) Otherwise, then the result of BV is the visible catalog object in the current working schema whose name is BV .

- 2) If the <parameter> P is specified, then

Case:

- a) If there is a request parameter RP with parameter name PN in the current request context, then the result of P is the parameter value of RP .
- b) Otherwise, the result of P is the parameter value of the session parameter with parameter name PN in the current session context.

Conformance Rules

None.

16.4 <type signature>

** Editor's Note (number 138) **

WG3:W05-020 suggests that discussion of this Subclause is needed to establish consensus. See Possible Problem [GQL-076](#).

Function

Specifies a type signature.

Format

```
<of type signature> ::=  
  [ <of type> ] <type signature>  
  
<type signature> ::=  
  <parenthesized formal parameter list> [ <of type> ] <procedure result type>  
  
<parenthesized formal parameter list> ::=  
  <left paren> [ <formal parameter list> ] <right paren>  
  
<formal parameter list> ::=  
  <mandatory formal parameter list>  
    [ <comma> <optional formal parameter list> ]  
  | <optional formal parameter list>  
  
<mandatory formal parameter list> ::=  
  <formal parameter declaration list>  
  
<optional formal parameter list> ::=  
  OPTIONAL <formal parameter definition list>  
  
<formal parameter declaration list> ::=  
  <formal parameter declaration> [ { <comma> <formal parameter declaration> }... ]  
  
<formal parameter definition list> ::=  
  <formal parameter definition> [ { <comma> <formal parameter definition> }... ]  
  
<formal parameter declaration> ::=  
  <parameter cardinality> <compact variable declaration>  
  
<formal parameter definition> ::=  
  <parameter cardinality> <compact variable definition>  
  
<parameter cardinality> ::=  
  SINGLE | MULTI | MULTIPLE  
  
<procedure result type> ::=  
  <value type>
```

Syntax Rules

None.

General Rules

- 1) If the <of type signature> OTS is specified, then the result of OTS is the result $ROTS$ of the <type signature> immediately contained in OTS and the type signature descriptor identified by OTS is the type signature descriptor of $ROTS$.
- 2) Let TS be the <type signature>.
- 3) Let $MFPL$ be the <mandatory formal parameter list> simply contained in TS .
- 4) Let $OFPL$ be the <optional formal parameter list> simply contained in TS .
- 5) Let RT be the <procedure result type> simply contained in TS .
- 6) The result of TS is a new type signature with a type signature descriptor comprising:
 - a) The required mandatory parameter declarations $MFPL$.
 - b) The allowed optional parameter definitions $OFPL$.
 - c) The result type RT .

** Editor's Note (number 139) **

Consider moving to SRs. See Possible Problem [GQL-076](#).

Conformance Rules

None.

16.5 <graph pattern>

Function

Specify a pattern to be matched in a graph.

Format

```
« WG3:BER-032R3 »

<graph pattern> ::= 
  [ <match mode> ] <path pattern list>
  [ <keep clause> ]
  [ <graph pattern where clause> ]
  [ <yield clause> ]

<match mode> ::= 
  <repeatable elements match mode>
  | <different edges match mode>

<repeatable elements match mode> ::= 
  REPEATABLE <element bindings or elements>

<different edges match mode> ::= 
  DIFFERENT <edge bindings or edges>

<element bindings or elements> ::= 
  ELEMENT [ BINDINGS ]
  | ELEMENTS

<edge bindings or edges> ::= 
  <edge synonym> [ BINDINGS ]
  | <edges synonym>

<path pattern list> ::= 
  <path pattern> [ { <comma> <path pattern> }... ] 

« WG3:BER-031 »

<path pattern> ::= 
  [ <path variable declaration> ] [ <path pattern prefix> ] <path pattern expression>

<path variable declaration> ::= 
  <path variable> <equals operator>

<keep clause> ::= 
  KEEP <path pattern prefix>

<graph pattern where clause> ::= 
  WHERE <search condition>
```

Syntax Rules

**** Editor's Note (number 140) ****

The following Syntax Rules are taken from SQL/PGQ and may require some revision to cater for the different context and additional options of GQL. See Possible Problem [GQL-164](#).

- 1) Let *GP* be the <graph pattern>.

**** Editor's Note (number 141) ****

Following sentence removed because <graph table> is not part of GQL: "Let GT be the <graph table> that simply contains GP ." See Possible Problem [GQL-164](#).

- 2) If BNF_1 and BNF_2 are instances of two BNF non-terminals, both contained in GP without an intervening <graph pattern>, then BNF_1 and BNF_2 are said to be *at the same depth of graph pattern matching*.

NOTE 96 — BNF_1 may contain BNF_2 while being at the same depth of graph pattern matching.

- 3) In a <path pattern list>, if two <path pattern>s expose an element variable EV , then both shall expose EV as an unconditional singleton variable.

NOTE 97 — This case expresses an implicit join on EV . Implicit joins between conditional singleton variables or group variables are forbidden.

« WG3:BER-031 »

- 4) Two <path pattern>s shall not expose the same subpath variable.

NOTE 98 — Implicit equijoins on subpath variables are not supported.

- 5) The name of a node variable shall not be equivalent to the name of an edge variable declared at the same depth of graph pattern matching.

- 6) If <keep clause> KP is specified, then:

a) Let PSP be the <path pattern prefix> simply contained in KP .

b) For each <path pattern> PP simply contained in GP :

i) PP shall not contain a <path search prefix>.

ii) Case:

« WG3:BER-031 »

1) If PP specifies a <path variable declaration>, let $PVDECL$ be that <path variable declaration>.

2) Otherwise, let $PVDECL$ be the zero-length character string.

iii) Case:

1) If PP specifies a <path mode prefix>, let PMP be that <path mode prefix>.

2) Otherwise, let PMP be the zero-length character string.

iv) Let PPE be the <path pattern expression> simply contained in PP .

v) PP is replaced by

$PVDECL \ PSP (PMP PPE)$

- c) The <keep clause> is removed from the <graph pattern>.

**** Editor's Note (number 142) ****

It has been suggested that it might be possible to treat the <path pattern prefix> specified in <keep clause> as merely providing a default <path pattern prefix> rather than a mandatory one for each <path pattern>. Whereas nested <path pattern prefix> is prohibited, this may be a feasible avenue of growth. On the other hand, perhaps a less definitive verb than KEEP may be appropriate when specifying a default <path pattern prefix>. See Language Opportunity [GQL-057](#).

- 7) After the preceding transformations, for every <path pattern> PP , if PP contains a <path pattern prefix> PPP that specifies a <path mode> PM , then:

- a) Case:

« WG3:BER-031 »

- i) If PP specifies a <path variable declaration>, let $PVDECL$ be that <path variable declaration>.
 - ii) Otherwise, let $PVDECL$ be the zero-length character string.
- b) Let PPE be the <path pattern expression> simply contained in PP .
- c) PP is replaced by

$PVDECL\ PPP\ (\ PM\ PPE\)$

NOTE 99 — One effect of the preceding transforms is that every <path mode> expressed outside a <parenthesized path pattern expression> is also expressed within a <parenthesized path pattern expression>. For example

ALL SHORTEST TRAIL GROUP <path pattern expression>
is rewritten as

« Email from: Fred Zemke, 2022-05-22 2231 »

ALL SHORTEST TRAIL GROUP (TRAIL <path pattern expression>)

The TRAIL specified outside the parentheses is now redundant. The benefit is that the definition of a consistent path binding in Subclause 21.2, “Machinery for graph pattern matching”, only has to consider <path mode>s declared in <parenthesized path pattern expression>s.

« WG3:BER-032R3 »

- 8) Let GPT be the <graph pattern> after the preceding syntactic transformations.
- 9) Let PPL be the <path pattern list> simply contained in GPT .
- 10) If GPT does not specify a <match mode>, then an implementation-defined (ID086) <match mode> is implicit.
- 11) Let MM be the <match mode> implicitly or explicitly specified by GPT .
- 12) If MM is <different edges match mode>, then:

- a) If PPL simply contains a <path pattern> that is selective, then PPL shall not simply contain any other <path pattern>.

NOTE 100 — If MM is <different edges match mode> and there is a selective <path pattern> SPP , then PPL must only contain SPP . If there is no selective <path pattern> in GPT , then there are no restrictions on how many non-selective <path pattern>s are contained in PPL . If MM is <repeatable elements match mode>, then there is no restriction on how many (selective and non-selective) <path pattern>s are contained in PPL .

- b) For every <path pattern> PP in PPL :

Case:

- i) If PP contains a <path pattern union> UA or a <path multiset alternation> UA that declares an edge variable EV , then:

- 1) If an operand in UA declares EV more than once, then EV is a *multiply-declared-different-edges-matched* edge variable.
- 2) If EV is also declared anywhere in PP outside of UA , then EV is a *multiply-declared-different-edges-matched* edge variable.

16.5 <graph pattern>

- ii) Otherwise, if *PP* declares an edge variable *EV* more than once, then *EV* is a *multiply-declared-different-edges-matched* edge variable.
- c) If two or more <path pattern>s in *PPL* expose an edge variable *EV*, then *EV* is a *multiply-declared-different-edges-matched* edge variable.

NOTE 101 — If *MM* is <different edges match mode>, then an edge variable *EV* is not a multiply-declared-different-edges-matched edge variable only if all of the following are true.

- i) *EV* appears in at most one <path pattern> in *GPT*.
- ii) *EV* appears in only one <edge pattern> within a <path pattern> *PP* in *GPT* if *PP* contains neither a <path pattern union> nor a <path multiset alternation>.
- iii) If *PP* contains either a <path pattern union> *UA* or a <path multiset alternation> *UA* that declares *EV*, then *EV* cannot appear anywhere else in *PP*. Furthermore, *EV* cannot appear more than once in each operand of *UA*.

- 13) Let *E* be an element variable declared by *GPT*.

Case:

- a) If *E* is exposed by *GPT* as an unconditional singleton, then *E* is a *global unconditional singleton* of *GPT*.

« WG3:BER-080 »

- b) Otherwise, let *PPPE* be the outermost <parenthesized path pattern expression> that exposes *E* as an unconditional singleton; the *unconditional singleton scope index* of *E* in *GPT* is the bracket index of *PPPE*.

NOTE 102 — Bracket index is defined in Subclause 21.2, “Machinery for graph pattern matching”. The unconditional singleton scope index is well-defined because implicit equijoins between conditional singleton variables or group variables are forbidden, hence there cannot be two <parenthesized path pattern expression>s that expose *E* as a conditional singleton or group variable unless one is contained in the other. For example,

```
( ( ( -[E]-> ) -[E]-> )* -[F]-> )*
```

The unconditional singleton scope of *E* is the middle <parenthesized path pattern expression> in the nest of three.

- 14) After the preceding transformations, every <quantified path primary> *QPP* contained in *GPT*, at least one of the following shall be true:

- a) The <graph pattern quantifier> of *QPP* is bounded.
- b) *QPP* is contained in a restrictive <parenthesized path pattern expression>.
- c) *QPP* is contained in a selective <path pattern>.
- d) *MM* is <different edges match mode>.

NOTE 103 — Provided neither the SIMPLE nor ACYCLIC keyword has been specified, this effectively imparts *TRAIL* semantics without requiring the presence of the *TRAIL* keyword.

« WG3:BER-031 »

« WG3:BER-032R3 »

- 15) Each <path variable> *PV* contained in *GPT* is the name of a *path variable*. The *degree of exposure* of *PV* and the path variable that it identifies is unconditional singleton.

- 16) If a <yield clause> is not specified, then:

« WG3:BER-032R3 »

- a) Let *EVIGS* be the set of the element variables in the scope of *GPT*.

- b) A <path variable> that is simply contained in *GPT* is a *path variable in the global scope of GPT*. Let *PVIGS* be the set of the path variables in the global scope of *GPT*.
- c) Let *GPVIGS* be a permutation of *EVIGS* \cup *PVIGS* in the order of their first occurrence as part of being immediately specified by an <element variable declaration> simply contained in *GP* or by a <path pattern> simply contained in *GPT* respectively.

**** Editor's Note (number 143) ****

Handling of subpath variables to be decided. See Language Opportunity [GQL-194](#).

- d) Let n be the number of graph pattern variables in *GPVIGS* and let $GPVIGS_i$, $1 \leq i \leq n$, be the i -th element of *GPVIGS*.
- e) Let the <yield item list> *YIL* be the comma-separated list of <yield item>s:

$GPVIGS_i \text{ AS } GPVIGS_i$
 $1 \leq i \leq n$.

- f) YIELD *YIL* is the implicit <yield clause> of *GP*.

**** Editor's Note (number 144) ****

The entire preceding General Rule was added in this document and is not included in SQL/PGQ. See Possible Problem [GQL-164](#).

General Rules

**** Editor's Note (number 145) ****

The following General Rules are taken from SQL/PGQ and may require some revision to cater for the different context and additional options of GQL. See Possible Problem [GQL-164](#).

- 1) Let *PG* be the current working graph.

**** Editor's Note (number 146) ****

The GR above was changed as the original was only applicable to SQL. See Possible Problem [GQL-164](#). "Let *GR* be the <graph reference> that is simply contained in *GT*. Let *SPG* be the graph that is identified by *GR*. The General Rules of Subclause 9.6, "Converting a tabular property graph to a pure property graph", in ISO/IEC 9075-16, are applied with *SPG* as *SQL-PROPERTY GRAPH*; let *PG* be the *PURE PROPERTY GRAPH* returned from the application of those General Rules."

« WG3:BER-032R3 adapted »

- 2) Let *PPL* be the <path pattern list> simply contained in *GPT*.
- 3) The General Rules of Subclause 21.2, "Machinery for graph pattern matching", are applied with *PG* as *PURE PROPERTY GRAPH* and *PPL* as *PATH PATTERN LIST*; let *MACH* be the *MACHINERY* returned from the application of those General Rules.
- 4) The following components of *MACH* are identified:
 - a) *ABC*, the alphabet, formed as the disjoint union of the following:

« WG3:BER-031 »

- i) *SVV*, the set of names of node variables.

- ii) SEV , the set of names of edge variables.
 - iii) SPS , the set of subpath symbols.
 - iv) SAS , the set of anonymous symbols.
 - v) SBS , the set of bracket symbols.
 - b) $REDUCE$, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.
- 5) Let NP be the number of <path pattern>s simply contained in PPL . Let PP_1, \dots, PP_{NP} be the <path pattern>s simply contained in PPL after the transformations in the Syntax Rules.
 «WG3:BER-032R3»
- 6) A multi-path binding $MPBINDING$ is *different-edges-matched* if for every edge binding $EB1 = (EV1, E)$ contained in $MPBINDING$;
- a) If $EV1$ is the anonymous edge symbol, then there is no other edge binding $EB2 = (EV2, E)$ contained in $MPBINDING$ that binds the edge E .
 - b) If $EV1$ is a global unconditional singleton variable, then there is no other edge binding $EB2 = (EV2, E)$ contained in $MPBINDING$ that binds E .
- 7) For every i , $1 \leq i \leq NP$:
- a) Let PPE be the <parenthesized path pattern expression> simply contained in PP_i .
 «WG3:BER-084R1»
 - b) The General Rules of Subclause 21.3, “Evaluation of a <path pattern expression>”, are applied with PG as *PURE PROPERTY GRAPH*, PPL as *PATH PATTERN LIST*, $MACH$ as *MACHINERY*, and PPE as *SPECIFIC BNF INSTANCE*; let $SMPPE_i$ be the *SET OF MATCHES* returned from the application of those General Rules.
 - c) Let $SMTEMP_i$ be a copy of $SMPPE_i$ where any binding containing an elementary binding $EB = (EV, E)$ where EV is a multiply-declared-restrictive-path-mode element variable has been removed.
- NOTE 104 — If an elementary variable has been multiply declared within a <parenthesized path pattern expression> PP that goes against the semantics inherent within TRAIL, SIMPLE, or ACYCLIC, then no matches are returned for PP .
- d) Case:
 «WG3:BER-032R3»
 - i) If PP_i is a selective <path pattern>, then
 - 1) Case:
 - A) If MM is <different edges match mode> then:
 - I) Let $SMDEP_i$ be the set of different-edges-matched multi-path bindings in $SMTEMP_i$.
 - II) Let $SMUP_i$ be a copy of $SMDEP_i$ where any binding containing an edge binding $EB = (EV, E)$ where EV is a multiply-declared-different-edges-matched edge variable has been removed.

NOTE 105 — If an edge variable has been multiply declared within a <path pattern> PP (respectively <graph pattern> GP) – as specified in *Syntax Rule*

12)b) (respectively [Syntax Rule 12\)c\)](#)) – then no matches are returned for PP (respectively GP).

- B) Otherwise, let $SMUP_i$ be $SMTEMP_i$.
 - 2) The General Rules of Subclause 21.4, “Evaluation of a selective <path pattern>”, are applied with PG as *PURE PROPERTY GRAPH*, PPL as *PATH PATTERN LIST*, $MACH$ as *MACHINERY*, PP_i as *SELECTIVE PATH PATTERN*, and $SMUP_i$ as *INPUT SET OF LOCAL MATCHES*; let SM_i be the *OUTPUT SET OF LOCAL MATCHES* returned from the application of those General Rules.
 - ii) Otherwise, let SM_i be $SMTEMP_i$.
- 8) Let $CROSS$ be the cross product $SM_1 \times \dots \times SM_{NP}$.
- « [WG3:BER-031](#) »
- 9) Let $INNER$ be the set of multi-path bindings MPB in $CROSS$ such that, for every unconditional singleton <element variable> USV exposed by PPL , USV is bound to a unique graph element by the elementary bindings of USV contained in MPB .

NOTE 106 — Anonymous symbols are not <element variable>s; there is no requirement that two anonymous symbols bind to the same graph element.

**** Editor's Note (number 147) ****

It has been proposed that unconditional singletons exposed by prior MATCH clauses may also be joined implicitly. See Language Opportunity [GQL-059](#).

- « [WG3:BER-032R3](#) »
- 10) Case:
- a) If MM is <different edges match mode> then:
 - i) Let $DEBINDINGS$ be the set of different-edges-matched multi-path bindings in $INNER$.
 - ii) Let $BINDINGS$ be a copy of $DEBINDINGS$ where any binding containing an edge binding $EB = (EV, E)$ where EV is a multiply-declared-different-edges-matched edge variable has been removed.
- NOTE 107 — If an edge variable has been multiply declared within a <path pattern> PP (respectively <graph pattern> GP) – as specified in [Syntax Rule 12\)b\)](#) (respectively [Syntax Rule 12\)c\)](#)) – then no matches are returned for PP (respectively GP).
- b) Otherwise, let $BINDINGS$ be $INNER$.
- 11) A *match* of GPT is a multi-path binding $M = (PB_1, \dots, PB_{NP})$ of NP path bindings in $BINDINGS$, such that the following are true:
- a) For every j , $1 \leq j \leq NP$, and for or every <parenthesized path pattern expression> $PPPE$ contained in PP_j , let i be the bracket index of $PPPE$, and let $'[i]$ and $']i'$ be the bracket symbols associated with $PPPE$. A *binding* of $PPPE$ is a substring of PB_j that begins with the bracket binding $('[i], '[i])$ and ends with the next bracket binding $(']i, ']i')$.

NOTE 108 — “Bracket index” is defined in Subclause 21.2, “Machinery for graph pattern matching”.

For every binding $BPPPE$ of $PPPE$ contained in PB_j , the following are true:

« [WG3:BER-031](#) »

- i) For every <element variable> EV that is exposed as an unconditional singleton by $PPPE$, EV is bound to a unique graph element by the element variable bindings contained in $BPPPE$.

NOTE 109 — Anonymous symbols are not <element variable>s; there is no requirement that two anonymous symbols bind to the same graph element.

« WG3:BER-032R3 »

- ii) If $PPPE$ contains a <parenthesized path pattern where clause> $PPPWC$, then the value of $V1$ is *True* when the General Rules of Subclause 21.5, “*Applying bindings to evaluate an expression*”, are applied with GPT as *GRAPH PATTERN*, the <search condition> simply contained in $PPPWC$ as *EXPRESSION*, $MACH$ as *MACHINERY*, M as *MULTI-PATH BINDING*, and a reference to $BPPPE$ as *REFERENCE TO LOCAL CONTEXT*; let $V1$ be the *VALUE* returned from the application of those General Rules.

** Editor's Note (number 148) **

The above GR is amended in GQL to specify the scope of path binding application. See Possible Problem [GQL-164](#).

- b) If GPT contains a <graph pattern where clause> $GPWC$, then the value of $V2$ is *True* when the General Rules of Subclause 21.5, “*Applying bindings to evaluate an expression*”, are applied with GPT as *GRAPH PATTERN*, $GPWC$ as *EXPRESSION*, $MACH$ as *MACHINERY*, M as *MULTI-PATH BINDING*, and a reference to M as *REFERENCE TO LOCAL CONTEXT*; let $V2$ be the *VALUE* returned from the application of those General Rules.
- 12) A reduced match $RM = (RPB_1, \dots, RPB_{NP})$ is obtained from a match $M = (PB_1, \dots, PB_{NP})$ as $RM = REDUCE(M)$.
- 13) The match records $MATCHES_{RM}$ are obtained for a reduced match $RM = \{ RPB_1, \dots, RPB_{NP} \}$ of GP by applying the General Rules of Subclause 16.16, “<*yield clause*>”, for YC when the reduced path bindings of RM are applied in the scope of GP in a new child execution context, as specified by General Rule 16) of Subclause 21.2, “*Machinery for graph pattern matching*”; $MATCHES_{RM}$ is the *YIELD* returned from the application of these General Rules.

** Editor's Note (number 149) **

The entire preceding General Rule was added in this document and is not included in SQL/PGQ. See Possible Problem [GQL-164](#).

- 14) Let $MATCHES$ be a new binding table comprising the match records obtained for each reduced match of GP in an implementation-dependent ([US004](#)) order.

** Editor's Note (number 150) **

The preceding General Rule was changed in this document and differs from the corresponding General Rule in SQL/PGQ. See Possible Problem [GQL-164](#).

- 15) The result of the application of the General Rules of this Subclause is $MATCHES$.

** Editor's Note (number 151) **

The preceding General Rule was changed in this document and differs from the corresponding General Rule in SQL/PGQ. See Possible Problem [GQL-164](#).

Conformance Rules

« WG3:BER-030 »

1) Without Feature G000, “Graph pattern”, conforming SQL language shall not contain a <graph pattern>.

« WG3:BER-032R3 »

- 2) Without Feature G001, “Repeatable-elements match mode”, conforming SQL language shall not contain a <repeatable elements match mode>.
- 3) Without Feature G002, “Different-edges match mode”, conforming SQL language shall not contain a <different edges match mode>.
- 4) Without Feature G003, “Explicit REPEATABLE ELEMENTS keyword”, conforming SQL language shall not contain a <match mode> that specifies REPEATABLE ELEMENTS or REPEATABLE ELEMENT BINDINGS.
- 5) Without Feature G004, “Path variables”, conforming SQL language shall not contain a <path pattern> that simply contains a <path variable declaration>.
- 6) Without Feature G005, “Path search prefix in a path pattern”, conforming SQL language shall not contain a <path pattern> that simply contains a <path pattern prefix> that is a <path search prefix>.
- 7) Without Feature G006, “Graph Pattern KEEP clause: path mode prefix”, conforming SQL language shall not contain a <keep clause>.
- 8) Without Feature G007, “Graph Pattern KEEP clause: path search prefix”, conforming SQL language shall not contain a <keep clause> that simply contains a <path search prefix>.
- 9) Without Feature G008, “Graph Pattern Where”, conforming SQL language shall not contain a <graph pattern where clause>.

« WG3:BER-030 »

16.6 <path pattern prefix>

Function

Specify a path-finding operation and a path mode.

Format

```
<path pattern prefix> ::=  
    <path mode prefix>  
    | <path search prefix>  
  
<path mode prefix> ::=  
    <path mode> [ <path or paths> ]  
  
<path mode> ::=  
    WALK  
    | TRAIL  
    | SIMPLE  
    | ACYCLIC  
  
<path search prefix> ::=  
    <all path search>  
    | <any path search>  
    | <shortest path search>
```

**** Editor's Note (number 152) ****

The ability to specify "cheapest" queries (analogous to SHORTEST, but minimizing the sum of costs along a path) is desirable. See [Language Opportunity GQL-052](#).

```
<all path search> ::=  
    ALL [ <path mode> ] [ <path or paths> ]  
  
<path or paths> ::=  
    PATH | PATHS  
  
<any path search> ::=  
    ANY [ <number of paths> ] [ <path mode> ] [ <path or paths> ]  
  
<number of paths> ::=  
    <unsigned integer specification>
```

**** Editor's Note (number 153) ****

This differs from the SQL/PGQ definition of <number of paths>.

```
<shortest path search> ::=  
    <all shortest path search>  
    | <any shortest path search>  
    | <counted shortest path search>  
    | <counted shortest group search>  
  
<all shortest path search> ::=  
    ALL SHORTEST [ <path mode> ] [ <path or paths> ]  
  
<any shortest path search> ::=  
    ANY SHORTEST [ <path mode> ] [ <path or paths> ]  
  
<counted shortest path search> ::=
```

```

SHORTEST <number of paths> [ <path mode> ] [ <path or paths> ]

<counted shortest group search> ::==
    SHORTEST <number of groups> [ <path mode> ] [ <path or paths> ] { GROUP | GROUPS }

<number of groups> ::==
    <unsigned integer specification>

```

**** Editor's Note (number 154) ****

This differs from the SQL/PGQ definition of <number of groups>.

**** Editor's Note (number 155) ****

In addition to SHORTEST GROUP, it has been proposed to support SHORTEST [*k*] WITH TIES, with the semantics to return the first *k* matches (where *k* defaults to 1) when sorting matches in ascending order on number of edges, and also return every match that has the same number of edges as the last of the *k* matches. This is the semantics of WITH TIES in Subclause 7.17, "<query expression>" in SQL/Foundation. See Language Opportunity [GQL-053](#).

Syntax Rules

- 1) If a <parenthesized path pattern expression> does not specify a <path mode prefix>, then WALK PATHS is implicit.
- 2) If a <path pattern prefix> *PPP* does not specify <all path search>, then:
 - a) Case:
 - i) If *PPP* does not simply contain a <path mode>, then let *PM* be WALK.
 - ii) Otherwise, let *PM* be the <path mode> simply contained in *PPP*.
 - b) Case:
 - i) If *PPP* does not simply contain a <number of paths> or <number of groups>, then let *N* be an <unsigned integer> whose value is 1 (one).
 - ii) Otherwise, let *N* be the <number of paths> or <number of groups> simply contained in *PPP*. The declared type of *N* shall be exact numeric with scale 0 (zero). If *N* is a <literal>, then the value of *N* shall be positive.
 - c) Case:
 - i) If *PPP* is an <any path search>, then *PPP* is equivalent to:
 ANY *N PM PATHS*
 - ii) If *PPP* is a <shortest path search>, then
 Case:
 - 1) If *PPP* is <all shortest path search>, then *PPP* is equivalent to:
 SHORTEST 1 *PM GROUP*
 - 2) If *PPP* is <any shortest path search>, then *PPP* is equivalent to:
 SHORTEST 1 *PM PATH*
 - 3) If *PPP* is <counted shortest path search>, then *PPP* is equivalent to:

16.6 <path pattern prefix>

SHORTEST N PM PATHS

- 4) If PPP is <counted shortest group search>, then PPP is equivalent to:

SHORTEST N PM GROUPS

- 3) A <path pattern prefix> that specifies a <path mode> other than WALK is *restrictive*. A <parenthesized path pattern expression> that immediately contains a restrictive <path mode prefix> is *restrictive*.
 « WG3:BER-084R1 deleted three Editor's Notes. »
 « WG3:BER-084R1 »

- 4) Let $PPPER$ be a restrictive <parenthesized path pattern expression>.

Case:

- a) If $PPPER$ simply contains a <path pattern union> UA or <path multiset alternation> UA that declares an edge variable EV , then:
 - i) If an operand in UA declares EV more than once, then EV is a *multiply-declared-restrictive-path-mode* edge variable.
 - ii) If EV is also declared anywhere in $PPPER$ outside of UA , then EV is a *multiply-declared-restrictive-path-mode* edge variable.
- b) If $PPPER$ declares an edge variable EV more than once, then EV is a multiply-declared-restrictive-path-mode edge variable.

NOTE 110 — An edge variable EV is a multiply-declared-restrictive-path-mode edge variable if it appears more than once in a restrictive path pattern (TRAIL, SIMPLE, or ACYCLIC).

- c) Otherwise,

Case:

- i) If <path mode> is SIMPLE then:

Case:

- 1) If $PPPER$ immediately contains a <path pattern expression> that immediately contains either a <path pattern union> UA or <path multiset alternation> UA that declares a vertnodeex variable VV , and an operand in UA declares VV more than once in any position other than in the first and last <node pattern>, then VV is a *multiply-declared-restrictive-path-mode* node variable.
- 2) If $PPPER$ simply contains a <path pattern union> UA or <path multiset alternation> UA that declares a node variable VV , then:
 - A) If an operand in UA declares VV more than once, then VV is a *multiply-declared-restrictive-path-mode* node variable.
 - B) If VV is also declared anywhere in $PPPER$ outside of UA , then VV is a *multiply-declared-restrictive-path-mode* node variable.
- 3) Otherwise, if $PPPER$ declares a node variable VV more than once so that VV appears in any position other than in the first and last <node pattern>, then VV is a *multiply-declared-restrictive-path-mode* node variable.

NOTE 111 — A node variable VV is a *multiply-declared-restrictive-path-mode* node variable if it appears more than once in any position other than in the first and last <node pattern> in a <parenthesized path pattern expression> with a <path mode> of SIMPLE.

- ii) If <path mode> is ACYCLIC then:

Case:

- 1) If *PPPER* simply contains a <path pattern union> *UA* or <path multiset alternation> *UA* that declares a node variable *VV*, then:
 - A) If an operand in *UA* declares *VV* more than once, then *VV* is a *multiply-declared-restrictive-path-mode* node variable.
 - B) If *VV* is also declared anywhere in *PPPER* outside of *UA*, then *VV* is a *multiply-declared-restrictive-path-mode* node variable.
- 2) Otherwise, if *PPPER* declares a node variable *VV* more than once, then *VV* is a *multiply-declared-restrictive-path-mode* node variable.

NOTE 112 — A node variable *VV* is a *multiply-declared-restrictive-path-mode* node variable if it appears more than once in a <parenthesized path pattern expression> with a <path mode> of ACYCLIC.

- 5) A <path search prefix> other than <all path search> is *selective*. A <path pattern> that simply contains a selective <path search prefix> is *selective*.

« WG3:BER-080 »

- 6) Let *PPPE* be a selective <path pattern>.
 - a) An element variable exposed by *PPPE* is an *interior variable* of *PPPE*.
 - b) A node variable *LVV* is the *left boundary variable* of *PPPE* if the following conditions are true:
 - i) *PPPE* exposes *LVV* as an unconditional singleton variable.
 - ii) *LVV* is declared in the first implicit or explicit <node pattern> *LVP* contained in *PPPE*.
 - iii) *LVP* is not contained in a <path pattern union> or <path multiset alternation> that is contained in *PPPE*.
 - c) A node variable *RVV* is the *right boundary variable* of *PPPE* if the following conditions are true:
 - i) *PPPE* exposes *RVV* as an unconditional singleton variable.
 - ii) *RVV* is declared in the last implicit or explicit <node pattern> *RVP* contained in *PPPE*.
 - iii) *RVP* is not contained in a <path pattern union> or <path multiset alternation> that is contained in *PPPE*.

**** Editor's Note (number 156) ****

With more work, it is possible to recognize when a node variable is declared uniformly in the first or the last position in every operand of a <path pattern union>. However, WG3:W04-009R1 declined to make the effort because it is easy for the user to factor out such a node pattern. For example, instead of

(X) -> (Y) | (X) -> (Z)

the user can write

(X) (-> (Y) | -> (Z))

Thus a more general definition of right or left boundary variable is possible. See Language Opportunity **GQL-056**.

- d) An element variable that is exposed by *PPPE* that is neither a left boundary variable of *PPPE* nor a right boundary variable of *PPPE* is a *strict interior variable* of *PPPE*.
- 7) An element variable that is not declared in a selective <path pattern> is an *exterior variable*.

16.6 <path pattern prefix>

- 8) A strict interior variable of one selective <path pattern> shall not be equivalent to an exterior variable, nor to an interior variable of another selective <path pattern>.

NOTE 113 — Implicit joins of boundary variables of selective <path pattern>s with exterior variables or boundary variables of other selective <path pattern>s are permitted.

- 9) A selective <path pattern> *SPP* shall not contain a reference to a graph pattern variable that is not declared by *SPP*.

NOTE 114 — This rule, and the prohibition of implicit joins to exterior variables and interior variables of other selective <path pattern>s, insure that each selective <path pattern> may be evaluated in isolation from any other <path pattern>.

General Rules

None.

NOTE 115 — Restrictive <path mode>s are enforced as part of the check for consistent path bindings in the generation of the set of local matches in Subclause 16.7, “<path pattern expression>”. Selective <path pattern>s are evaluated by Subclause 21.4, “Evaluation of a selective <path pattern>”.

Conformance Rules

« WG3:BER-030 »

- 1) Without Feature G010, “Explicit WALK keyword”, conforming SQL language shall not contain a <path mode> that specifies WALK.
- 2) Without Feature G011, “Advanced Path Modes: TRAIL”, conforming SQL language shall not contain a <path mode> that specifies TRAIL.
- 3) Without Feature G012, “Advanced Path Modes: SIMPLE”, conforming SQL language shall not contain a <path mode> that specifies SIMPLE.
- 4) Without Feature G013, “Advanced Path Modes: ACYCLIC”, conforming SQL language shall not contain a <path mode> that specifies ACYCLIC.
- 5) Without Feature G014, “Explicit PATH/PATHS keywords”, conforming SQL language shall not contain a <path or paths>.
- 6) Without Feature G015, “All path search: explicit ALL keyword”, conforming SQL language shall not contain an <all path search>.
- 7) Without Feature G016, “Any path search”, conforming SQL language shall not contain an <any path search>.
- 8) Without Feature G017, “All shortest path search”, conforming SQL language shall not contain a <shortest path search>.
- 9) Without Feature G018, “Any shortest path search”, conforming SQL language shall not contain an <any shortest path search>.
- 10) Without Feature G019, “Counted shortest path search”, conforming SQL language shall not contain a <counted shortest path search>.
- 11) Without Feature G020, “Counted shortest group search”, conforming SQL language shall not contain a <counted shortest group search>.

16.7 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```

<path pattern expression> ::=

  <path term>
  | <path multiset alternation>
  | <path pattern union>

<path multiset alternation> ::=

  <path term> <multiset alternation operator> <path term>
  [ { <multiset alternation operator> <path term> }... ]

<path pattern union> ::=

  <path term> <vertical bar> <path term> [ { <vertical bar> <path term> }... ]

<path term> ::=

  <path factor>
  | <path concatenation>

<path concatenation> ::=

  <path term> <path factor>

<path factor> ::=

  <path primary>
  | <quantified path primary>
  | <questioned path primary>

<quantified path primary> ::=

  <path primary> <graph pattern quantifier>

<questioned path primary> ::=

  <path primary> <question mark>

  NOTE 116 — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {0,1}: the
  quantifier {0,1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes
  to group. However, <question mark> does expose any singleton variables as conditional singletons.

<path primary> ::=

  <element pattern>
  | <parenthesized path pattern expression>
  | <simplified path pattern expression>

<element pattern> ::=

  <node pattern>
  | <edge pattern>

<node pattern> ::=

  <left paren> <element pattern filler> <right paren>

<element pattern filler> ::=

  [ <element variable declaration> ]
  [ <is label expression> ]
  [ <element pattern predicate> ]

<element variable declaration> ::=

  <element variable>

```

16.7 <path pattern expression>

```

<is label expression> ::= 
  <is or colon> <label expression>

<is or colon> ::= 
  IS
  | <colon>

<element pattern predicate> ::= 
  <element pattern where clause>
  | <element property specification>

<element pattern where clause> ::= 
  WHERE <search condition>

<element property specification> ::= 
  <left brace> <property key value pair list> <right brace>

<property key value pair list> ::= 
  <property key value pair> [ { <comma> <property key value pair> }... ]

<property key value pair> ::= 
  <property name> <colon> <value expression>

<edge pattern> ::= 
  <full edge pattern>
  | <abbreviated edge pattern>

<full edge pattern> ::= 
  <full edge pointing left>
  | <full edge undirected>
  | <full edge pointing right>
  | <full edge left or undirected>
  | <full edge undirected or right>
  | <full edge left or right>
  | <full edge any direction>

<full edge pointing left> ::= 
  <left arrow bracket> <element pattern filler> <right bracket minus>

<full edge undirected> ::= 
  <tilde left bracket> <element pattern filler> <right bracket tilde>

<full edge pointing right> ::= 
  <minus left bracket> <element pattern filler> <bracket right arrow>

<full edge left or undirected> ::= 
  <left arrow tilde bracket> <element pattern filler> <right bracket tilde>

<full edge undirected or right> ::= 
  <tilde left bracket> <element pattern filler> <bracket tilde right arrow>

<full edge left or right> ::= 
  <left arrow bracket> <element pattern filler> <bracket right arrow>

<full edge any direction> ::= 
  <minus left bracket> <element pattern filler> <right bracket minus>

```

** Editor's Note (number 157) **

In the BNF for <full edge any direction>, the delimiter tokens <~[]~> have been suggested as a synonym for -[]- as part of Feature GA10, "Undirected edge patterns". The synonym for the <abbreviated edge pattern> - (<minus sign>) would then be <~>, the synonym for <simplified defaulting any direction> would use the delimiter tokens <~/ /~> and the synonym for <simplified override any direction> would use the tokens <~ and > surrounding

a label as originally proposed in WG3:MMX-060. These synonyms might be considered to make the table of edge patterns more harmonious and internally consistent. See [Language Opportunity \[GQL-212\]](#).

```
<abbreviated edge pattern> ::=  

    <left arrow>  

    | <tilde>  

    | <right arrow>  

    | <left arrow tilde>  

    | <tilde right arrow>  

    | <left minus right>  

    | <minus sign>
```

**** Editor's Note (number 158) ****

The syntax for <abbreviated edge pattern> is incompatible with syntax in the widely used graph query language Cypher. See [Possible Problem \[GQL-251\]](#).

```
<parenthesized path pattern expression> ::=  

    <left paren>  

        [ <subpath variable declaration> ]  

        [ <path mode prefix> ]  

        <path pattern expression>  

        [ <parenthesized path pattern where clause> ]  

    <right paren>  
  

<subpath variable declaration> ::=  

    <subpath variable> <equals operator>  
  

<parenthesized path pattern where clause> ::=  

    WHERE <search condition>
```

Syntax Rules

« [WG3:BER-025](#) »

- 1) Let *RIGHTMINUS* be the following collection of <token>s: <right bracket minus>, <left arrow>, <slash minus>, and <minus sign>.
 NOTE 117 — These are the tokens `]-`, `<-`, `/-`, and `-`, which expose a minus sign on the right.
- 2) Let *LEFTMINUS* be the following collection of <token>s: <minus left bracket>, <right arrow>, <minus slash>, and <minus sign>.
 NOTE 118 — These are the tokens `-[`, `->`, `-/`, and `-`, which expose a minus sign on the left. <minus sign> itself is in both *RIGHTMINUS* and *LEFTMINUS*.
- 3) A <path pattern expression> shall not juxtapose a <token> from *RIGHTMINUS* followed by a <token> from *LEFTMINUS* without a <separator> between them.
 NOTE 119 — Otherwise the concatenation of the two tokens would include the sequence of two <minus sign>s, which is a <simple comment introducer>.
- 4) A <path pattern expression> that contains at the same depth of graph pattern matching a variable quantifier, a <questioned path primary>, a <path multiset alternation>, or a <path pattern union> is a *possibly variable length path pattern*.
- 5) A <path pattern expression> that is not a possibly variable length path pattern is a *fixed length path pattern*.

« [WG3:BER-030 deleted one rule and associated Editor's Note.](#) »

16.7 <path pattern expression>

- 6) The *minimum path length* of certain BNF non-terminals defined in this Subclause is defined recursively as follows:
- The minimum path length of a <node pattern> is 0 (zero).
 - The minimum path length of an <edge pattern> is 1 (one).
 - The minimum path length of a <path concatenation> is the sum of the minimum path lengths of its operands.
 - The minimum path length of a <path pattern union> or <path multiset alternation> is the minimum of the minimum path length of its operands.
 - The minimum path length of a <quantified path primary> is the product of the minimum path length of the simply contained <path primary> and the value of the <lower bound>.
 - The minimum path length of a <questioned path primary> is 0 (zero).
 - The minimum path length of a <parenthesized path pattern expression> is the minimum path length of the simply contained <path pattern expression>.
 - If $BNT1$ and $BNT2$ are two BNF non-terminals such that $BNT1 ::= BNT2$ and the minimum path length of $BNT2$ is defined, then the minimum path length of $BNT1$ is also defined and is the same as the minimum path length of $BNT2$.
- 7) The <path primary> immediately contained in a <quantified path primary> or <questioned path primary> shall have minimum path length that is greater than 0 (zero).
- 8) The <path primary> simply contained in a <quantified path primary> shall not contain a <quantified path primary> at the same depth of graph pattern matching.

**** Editor's Note (number 159) ****

It may be possible to permit nested quantifiers. WG3:W01-014 contained a discussion of a way to support aggregates at different depths of aggregation if there are nested quantifiers. See Language Opportunity [GQL-036](#).

- 9) Let PMA be a <path multiset alternation>.
- A <path term> simply contained in PMA is a *multiset alternation operand* of PMA .
 - Let $NOPMA$ be the number of multiset alternation operands of PMA . Let $OPMA_1, \dots, OPMA_{NOPMA}$ be an enumeration of the operands of PMA .
 - Any <subpath variable>s declared by <subpath variable declaration>s simply contained in the multiset alternation operands of PMA shall be mutually distinct.
 - Let $SOPMA_1, \dots, SOPMA_{NOPMA}$ be implementation-dependent ([UV008](#)) <identifier>s that are mutually distinct and distinct from every <element variable>, <subpath variable> and <path variable> contained in GP .
 - For every i , $1 \text{ (one)} \leq i \leq NOPMA$.
- Case:
- If $OPMA_i$ is a <parenthesized path pattern expression> that simply contains a <subpath variable declaration>, then let $OPMAX_i$ be $OPMA_i$.
 - Otherwise, let $OPMAX_i$ be the <parenthesized path pattern expression>

$(SOPMA_i = OPMA_i)$

- f) *PMA* is equivalent to:

$OPMAX_1 \mid \dots \mid OPMAX_{NOPMA}$

« WG3:BER-031 »

- 10) A <path term> *PPUOP* simply contained in a <path pattern union> *PSD* is a *path pattern union operand* of *PSD*.

**** Editor's Note (number 160) ****

Path pattern union is not defined using left recursion. WG3:SXM-052 believed that it should be possible to support left recursion but declined to do so for expediency. It is a Language Opportunity to support left recursion. See [Language Opportunity GQL-025](#).

« WG3:BER-031 »

PPUOP shall not contain a reference to an element variable that is not declared in *PPUOP* or outside of *PSD*.

- 11) An <element pattern> *EP* that contains an <element pattern where clause> *EPWC* is transformed as follows:

- Let *EPF* be the <element pattern filler> simply contained in *EP*.
- Let *PREFIX* be the <delimiter token> contained in *EP* before *EPF* and let *SUFFIX* be the <delimiter token> contained in *EP* after *EPF*.
- Let *EV* be the <element variable> simply contained in *EPF*. Let *ILE* be the <is label expression> contained in *EPF*, if any, otherwise let *ILE* be the zero-length string.
- EP* is replaced by

(*PREFIX EV ILE SUFFIX EPWC*)

- 12) An <element pattern> that does not contain an <element variable declaration>, an <is label expression>, or an <element pattern predicate> is said to be *empty*.

- 13) Each <path pattern expression> is transformed in the following steps:

- a) If the <path primary> immediately contained in a <quantified path primary> or <questioned path primary> is an <edge pattern> *EP*, then *EP* is replaced by

(*EP*)

NOTE 120 — For example:

-> *

becomes:

(->) {0,}

which in later transformations becomes:

(()) -> (()) {0,}

- b) If two successive <element pattern>s contained in a <path concatenation> at the same depth of graph pattern matching are <edge pattern>s, then an implicit empty <node pattern> is inserted between them.
- c) If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not preceded by a <node pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <node pattern> *VP* is inserted in *PST* immediately prior to *EP*.

16.7 <path pattern expression>

- d) If an edge pattern EP contained in a <path term> PST at the same depth of graph pattern matching is not followed by a <node pattern> contained in PST at the same depth of graph pattern matching, than an implicit empty <node pattern> VP is inserted in PST immediately after EP .

NOTE 121 — As a result of the preceding transformations, a fixed length path pattern has an odd number of <element pattern>s, beginning and ending with <node pattern>s, and alternating between <node pattern>s and <edge pattern>s.

- e) Every <abbreviated edge pattern> AEP is replaced with an empty <full edge pattern> as follows

Case:

- i) If AEP is <left arrow>, then AEP is replaced by

$<- [] ->$

- ii) If AEP is <tilde>, then AEP is replaced by

$\sim [] \sim$

- iii) If AEP is <right arrow>, then AEP is replaced by

$- [] ->$

- iv) If AEP is <left arrow tilde>, then AEP is replaced by

$<\sim [] \sim$

- v) If AEP is <tilde right arrow>, then AEP is replaced by

$\sim [] \sim ->$

- vi) If AEP is <left minus right>, then AEP is replaced by

$<- [] ->$

- vii) If AEP is <minus sign>, then AEP is replaced by

$- [] -$

- 14) The *minimum node count* of certain BNF non-terminals defined in this Subclause is defined recursively as follows:

- a) The minimum node count of a <node pattern> is 1 (one).

- b) The minimum node count of an <edge pattern> is 0 (zero).

« WG3:BER-071 »

- c) The minimum node count of a <path concatenation> PC is:

Case:

- i) If two successive <element pattern>s contained in PC at the same depth of graph pattern matching are <node pattern>s, then 1 (one) less than the sum of the minimum node counts of its operands.

- ii) Otherwise, the sum of the minimum node counts of its operands.

- d) The minimum node count of a <path pattern union> or <path multiset alternation> is the minimum of the minimum node count of its operands.

- e) The minimum node count of a <quantified path primary> is the product of the minimum node count of the simply contained <path primary> and the value of the <lower bound> of the simply contained <graph pattern quantifier>.
- f) The minimum node count of a <questioned path primary> is 0 (zero).
- g) The minimum node count of a <parenthesized path pattern expression> is the minimum node count of the simply contained <path pattern expression>.

« WG3:BER-071 »

- h) If $BNF1$ and $BNF2$ are two BNF non-terminals such that $BNF1 ::= BNF2$ and the minimum node count of $BNF2$ is defined, then the minimum node count of $BNF1$ is also defined and is the same as the minimum node count of $BNF2$.
- 15) The <path pattern expression> simply contained in a <path pattern> shall have a minimum node count that is greater than 0 (zero).

NOTE 122 — The minimum node count is computed after the syntactic transform that adds implicit node patterns.
 Thus a single <edge pattern> is a permitted <path pattern> because it implies two <node pattern>s.

« WG3:BER-031 »

- 16) An <element variable> EV contained in an <element variable declaration> $GPVD$ is said to be *declared* by $GPVD$, and by the <element pattern> EP that simply contains $GPVD$. The <element variable> is the name of an element variable, which is also declared by $GPVD$ and EP . EV is a *primary variable*.
- 17) An element variable that is declared by a <node pattern> is a *node variable*. An element variable that is declared by an <edge pattern> is an *edge variable*.
- 18) The scope of an <element variable> that is declared by an <element pattern> EP includes the following:
 - a) The <element pattern where clause> of EP , if any.
 - b) The <graph pattern where clause> of GP , if any.
 - c) The <parenthesized path pattern where clause> of a <parenthesized path pattern expression> that contains EP at the same depth of graph pattern matching.
 - d) The explicit or implicit <yield clause> of GP .

**** Editor's Note (number 161) ****

This item is changed in this document and differs from the corresponding item in SQL/PGQ.

**** Editor's Note (number 162) ****

This item was changed because <graph table columns clause> is not part of GQL: “The <graph table columns clause> of GT ” but has been replaced with an approximate GQL equivalent that requires further revision. See Possible Problem [GQL-164](#).

**** Editor's Note (number 163) ****

The set of variables in scope for the <graph pattern where clause> and the <yield clause> needs to be reviewed.
 See Possible Problem [GQL-172](#).

« WG3:BER-031 »

- 19) A <subpath variable> SV contained in a <subpath variable declaration> SVD is said to be *declared* by SVD , and by the <parenthesized path pattern expression> $PPPE$ that simply contains SVD . SV is the name of a subpath variable, which is also declared by SVD and $PPPE$.

16.7 <path pattern expression>

- 20) If *EP* is an <element pattern> that contains an <element pattern where clause> *EPWC*, then *EP* shall simply contain an <element variable declaration> *GPVD*.

« WG3:RKE-031 »

- 21) If *EV* is an element variable or subpath variable, and *BNT* is an instance of a BNF non-terminal, then the terminology “*BNT* exposes *EV*” is defined as follows. The full terminology is one of the following: “*BNT* exposes *EV* as an unconditional singleton variable”, “*BNT* exposes *EV* as a conditional singleton variable”, “*BNT* exposes *EV* as an effectively bounded group variable” or “*BNT* exposes *EV* as an effectively unbounded group variable”. The terms “unconditional singleton variable”, “conditional singleton variable”, “effectively bounded group variable”, and “effectively unbounded group variable” are called the *degree of exposure*.

- a) An <element pattern> *EP* that declares an element variable *EV* exposes *EV* as an unconditional singleton.

« WG3:BER-031 »

- b) A <parenthesized path pattern expression> *PPPE* that simply contains a <subpath variable declaration> that declares *EV* exposes *EV* as an unconditional singleton variable. *PPPE* shall not contain another <parenthesized path pattern expression> that declares *EV*.

- c) If a <path concatenation> *PPC* declares *EV* then let *PT* be the <path term> and let *PF* be the <path factor> simply contained in *PPC*.

Case:

« WG3:BER-031 »

- i) If *EV* is exposed as an unconditional singleton by both *PT* and *PF*, then *EV* is exposed as an unconditional singleton by *PPC*. *EV* shall not be a subpath variable.

NOTE 123 — This case expresses an implicit join on *EV* within *PPC*. Implicit joins between conditional singleton variables, group variables, or subpath variables are forbidden.

- ii) Otherwise, *EV* shall only be exposed by one of *PT* or *PF*. In this case *EV* is exposed by *PPC* in the same degree that it is exposed by *PT* or *PF*.

- d) If a <path pattern union> or <path multiset alternation> *PA* declares *EV*, then

Case:

- i) If every operand of *PA* exposes *EV* as an unconditional singleton variable, then *PA* exposes *EV* as an unconditional singleton variable.

- ii) If at least one operand of *PA* exposes *EV* as an effectively unbounded group variable, then *PA* exposes *EV* as an effectively unbounded group variable.

- iii) If at least one operand of *PA* exposes *EV* as an effectively bounded group variable, then *PA* exposes *EV* as an effectively bounded group variable.

- iv) Otherwise, *PA* exposes *EV* as a conditional singleton variable.

- e) If a <quantified path primary> *QPP* declares *EV*, then let *PP* be the <path primary> simply contained in *QPP*.

Case:

- i) If *QPP* contains a <graph pattern quantifier> that is a <fixed quantifier> or a <general quantifier> that contains an <upper bound> and *PP* does not expose *EV* as an effectively unbounded group variable, then *QPP* exposes *EV* as an effectively bounded group variable.

- ii) If *QPP* is contained at the same depth of graph pattern matching in a restrictive <parenthesized path pattern expression>, then *QPP* exposes *EV* as an effectively bounded group variable.

NOTE 124 — The preceding definition is applied after the syntactic transformation to insure that every <path mode prefix> is at the head of a <parenthesized path pattern expression>.
 - iii) Otherwise, *QPP* exposes *EV* as an effectively unbounded group variable.
- f) If a <questioned path primary> *QUPP* declares *EV*, then let *PP* be the <path primary> simply contained in *QUPP*.

Case:

- i) If *PP* exposes *EV* as a group variable, then *QUPP* exposes *EV* as a group variable with the same degree of exposure.
- ii) Otherwise, *QUPP* exposes *EV* as a conditional singleton variable.

« WG3:BER-031 »

- g) A <parenthesized path pattern expression> exposes the same variables as the simply contained <path pattern expression>, in the same degree of exposure.

NOTE 125 — A restrictive <path mode> declared by a <parenthesized path pattern expression> makes variables effectively bounded, but it does so even for proper subexpressions within the scope of the <path mode> and has already been handled by the rules for <quantified path primary>.
- h) If a <path pattern> *PP* declares *EV*, then let *PPE* be the simply contained <path pattern expression>.

Case:

 - i) If *PPE* exposes *EV* as an unconditional singleton, a conditional singleton, or an effectively bounded group variable, then *PP* exposes *EV* with the same degree of exposure.
 - ii) Otherwise, *PP* exposes *EV* as an effectively bounded group variable.

NOTE 126 — That is, even if *PPE* exposes *EV* as an effectively unbounded group variable, *PP* still exposes *EV* as effectively bounded, because in this case *PP* is required to be a selective <path pattern>.

**** Editor's Note (number 164) ****

WG3:W04-009R1 defined “effectively bounded group variable” but did not use the definition. The definition will be used when we define predicates on aggregates, at which time we will want a Syntax Rules stating that if a group variable *GV* is referenced in a WHERE clause, then it shall be effectively bounded and the reference shall be contained in an aggregated argument of an <aggregate function>. See Possible Problem [GQL-050](#).

« WG3:BER-031 »

- 22) If *BNT* is a BNF non-terminal that exposes a graph pattern variable *GPV* with a degree of exposure *DEGREE*, then *BNT* is also said to expose the name of *GPV* with degree of exposure *DEGREE*.
- 23) A <parenthesized path pattern where clause> *PPPWC* simply contained in a <parenthesized path pattern expression> *PPPE* shall not reference a path variable.

**** Editor's Note (number 165) ****

WG3:W04-009R1 recognized that a graph query may have a sequence of MATCH clauses, with the bindings of one MATCH clause *MC1* visible in all subsequent MATCH clauses in the same invocation of <graph table>, and that it should be permissible to reference such variables in any <parenthesized path pattern where clause> simply contained in a

subsequent MATCH clause *MC2*. The relevance of this LO to GQL needs to be investigated. See [Language Opportunity GQL-051](#).

General Rules

None.

NOTE 127 — The evaluation of a <path pattern expression> is performed by the General Rules of Subclause 21.3, “Evaluation of a <path pattern expression>”.

Conformance Rules

« [WG3:BER-030 deleted two CRs.](#) »
 « [WG3:BER-030](#) »

- 1) Without Feature G030, “Path Multiset Alternation”, conforming SQL language shall not contain a <path multiset alternation>.
- 2) Without Feature G031, “Path Multiset Alternation: variable length path operands”, in conforming SQL language, an operand of a <path multiset alternation> shall be a fixed length path pattern.
- 3) Without Feature G032, “Path Pattern Union”, conforming SQL language shall not contain a <path pattern union>.
- 4) Without Feature G033, “Path Pattern Union: variable length path operands”, in conforming SQL language, an operand of a <path pattern union> shall be a fixed length path pattern.
- 5) Without Feature G034, “Path concatenation”, conforming SQL language shall not contain a <path concatenation>.
- 6) Without Feature G035, “Quantified Paths”, conforming SQL language shall not contain a <quantified path primary> that does not immediately contain a <path primary> that is an <edge pattern>.
- 7) Without Feature G036, “Quantified Edges”, conforming SQL language shall not contain a <quantified path primary> that immediately contains a <path primary> that is an <edge pattern>.
- 8) Without Feature G037, “Questioned Paths”, conforming SQL language shall not contain a <questioned path primary>.
- 9) Without Feature G038, “Parenthesized path pattern expression”, conforming SQL language shall not contain a <parenthesized path pattern expression>.
- 10) Without Feature G039, “Simplified Path Pattern Expression: full defaulting”, conforming SQL language shall not contain a <simplified path pattern expression> that is not a <simplified defaulting left>, a <simplified defaulting right>, or a <simplified defaulting any direction>.
- 11) Without Feature G040, “Vertex pattern”, conforming SQL language shall not contain a <node pattern>.
- 12) Without Feature G041, “Non-local element pattern predicates”, in conforming SQL language, the <element pattern where clause> of an <element pattern> *EP* shall only reference the <element variable> declared in *EP*.
- 13) Without Feature G042, “Basic Full Edge Patterns”, conforming SQL language shall not contain a <full edge any direction>, a <full edge pointing left>, or a <full edge pointing right>.
- 14) Without Feature G043, “Complete Full Edge Patterns”, conforming SQL language shall not contain a <full edge pattern> that is not a <full edge any direction>, a <full edge pointing left>, or a <full edge pointing right>.

- 15) Without Feature G044, “Basic Abbreviated Edge Patterns”, conforming SQL language shall not contain an <abbreviated edge pattern> that is a <minus sign>, <left arrow>, or <right arrow>.
- 16) Without Feature G045, “Complete Abbreviated Edge Patterns”, conforming SQL language shall not contain an <abbreviated edge pattern> that is not a <minus sign>, <left arrow>, or <right arrow>.
- 17) Without Feature G046, “Relaxed topological consistency: Adjacent vertex patterns”, in conforming SQL language, between any two <node pattern>s contained in a <path pattern expression> there shall be at least one <edge pattern>, <left paren>, or <right paren>.
- 18) Without Feature G047, “Relaxed topological consistency: Concise edge patterns”, in conforming SQL language, any <edge pattern> shall be immediately preceded and followed by a <node pattern>.
- 19) Without Feature G048, “Parenthesized Path Pattern: Subpath variable declaration”, conforming SQL language shall not contain a <parenthesized path pattern expression> that simply contains a <subpath variable declaration>.
- 20) Without Feature G049, “Parenthesized Path Pattern: Path mode prefix”, conforming SQL language shall not contain a <parenthesized path pattern expression> that immediately contains a <path mode prefix>.
- 21) Without Feature G050, “Parenthesized Path Pattern: Where clause”, conforming SQL language shall not contain a <parenthesized path pattern where clause>.
- 22) Without Feature G051, “Parenthesized Path Pattern: Non-local predicates”, in conforming SQL language, a <parenthesized path pattern where clause> simply contained in a <parenthesized path pattern expression> *PPPE* shall not reference an <element variable> that is not declared in *PPPE*.

16.8 <simple graph pattern>

Function

Define a <simple graph pattern>.

Format

```
<simple graph pattern> ::=  
  <simple path pattern list>  
  
<simple path pattern list> ::=  
  <simple path pattern> [ { <comma> <simple path pattern> }... ]  
  
<simple path pattern> ::=  
  <path pattern expression>
```

Syntax Rules

- 1) A <simple path pattern> shall be a <path pattern> that after the application of the Syntax Rules for <path pattern> is either a <node pattern> or a concatenation of <node pattern> <edge pattern> <node pattern>.
- 2) Let *ECSPP* be a <simple path pattern> that simply contains an <edge pattern>.
 - a) Every <node pattern> simply contained in *ECSPP* shall simply contain an <element variable declaration>.
 - b) Every <node pattern> simply contained in *ECSPP* shall not simply contain an <is label expression>, or an <element pattern predicate>.
« Editorial: Consistently use regular identifier »
 - c) If any <regular identifier> simply contained in an <element variable declaration> that is simply contained in a <node pattern> simply contained in *ECSPP* references a variable that is not in scope, then an exception condition is raised: *syntax error or access rule violation — invalid reference (42002)*.
- 3) No <element pattern> simply contained in a <simple path pattern> shall simply contain an <element pattern where clause>.
- 4) Every <full edge pattern> simply contained in a <simple path pattern> after the application of the Syntax Rules for <abbreviated edge pattern>, shall simply contain one of <full edge pointing left>, <full edge undirected>, or <full edge pointing right>.
NOTE 128 — The Syntax Rules for <abbreviated edge pattern> rewrite an <abbreviated edge pattern> to a <full edge pattern>. See [Syntax Rule 13\)e](#).
- 5) No <is label expression> simply contained in a <simple path pattern> shall contain a <label disjunction> or a <label negation>.
- 6) No <value expression> simply contained in a <simple path pattern> *SPP* shall contain a <binding variable> that references a graph element that is introduced by an <element variable> simply contained in *SPP*.

General Rules

None.

NOTE 129 — <simple path pattern> provides restrictions for path pattern syntax used in other subclauses. The subclauses that use <simple path pattern> perform the evaluation of the resulting path pattern and so no general rules are needed in this subclause.

Conformance Rules

None.

« WG3:BER-030 moved one Subclause »

16.9 <label expression>

Function

Specify an expression that matches one or more labels of a graph.

Format

```
<label expression> ::=  
  <label term>  
  | <label disjunction>  
  
<label disjunction> ::=  
  <label expression> <vertical bar> <label term>  
  
<label term> ::=  
  <label factor>  
  | <label conjunction>  
  
<label conjunction> ::=  
  <label term> <ampersand> <label factor>  
  
<label factor> ::=  
  <label primary>  
  | <label negation>  
  
<label negation> ::=  
  <exclamation mark> <label primary>  
  
<label primary> ::=  
  <label name>  
  | <wildcard label>  
  | <parenthesized label expression>  
  
<wildcard label> ::=  
  <percent>  
  « WG3:BER-057 deleted one Editor's Note. »  
  
<parenthesized label expression> ::=  
  <left paren> <label expression> <right paren>
```

Syntax Rules

** Editor's Note (number 166) **

The following Syntax Rules are taken from SQL/PGQ and require some revision to cater for the different context and additional options of GQL. See Possible Problem [GQL-164](#).

** Editor's Note (number 167) **

The following SR (2) was changed as the original was only applicable to SQL since GQL does not have a <graph table> and does not use <graph reference>. Let *GP* be the <graph pattern> that simply contains *LE*. Let *GT* be the <graph table> that simply contains *GP*. Let *GR* be the <graph reference> that is simply contained in *GT*. Let *PG* be the graph that is identified by *GR*. See Possible Problem [GQL-164](#).

- 1) Let *LE* be the <label expression>.
- 2) Let *GP* be the <graph pattern> that simply contains *LE*. Let *PG* be the current working graph.

- 3) Every <label name> contained in *LE* shall identify a label of *PG*.
- 4) Case:
 - a) If *LE* is simply contained in a <node pattern>, then *LE* is a *node <label expression>*. Every <label name> contained in *LE* shall identify a node label of *PG*.
 - b) Otherwise, *LE* is an *edge <label expression>*. Every <label name> contained in *LE* shall identify an edge label of *PG*.

General Rules

None.

Conformance Rules

« WG3:BER-030 »

- 1) Without Feature G070, “Label expression: Label disjunction”, conforming SQL language shall not contain a <label disjunction>.
- 2) Without Feature G071, “Label expression: Label conjunction”, conforming SQL language shall not contain a <label conjunction>.
- 3) Without Feature G072, “Label expression: Label negation”, conforming SQL language shall not contain a <label negation>.
- 4) Without Feature G073, “Label expression: Individual label name”, conforming SQL language shall not contain a <label expression> that is a <label name>.
- 5) Without Feature G074, “Label expression: Wildcard label”, conforming SQL language shall not contain a <wildcard label>.
- 6) Without Feature G075, “Parenthesized label expression”, conforming SQL language shall not contain a <parenthesized label expression>.

« WG3:BER-030 »

16.10 <graph pattern quantifier>

Function

Specify a graph pattern quantifier.

Format

```

<graph pattern quantifier> ::==
  <asterisk>
  | <plus sign>
  | <fixed quantifier>
  | <general quantifier>

<fixed quantifier> ::==
  <left brace> <unsigned integer> <right brace>

<general quantifier> ::==
  <left brace> [ <lower bound> ] <comma> [ <upper bound> ] <right brace>

<lower bound> ::==
  <unsigned integer>

<upper bound> ::==
  <unsigned integer>

```

Syntax Rules

- 1) The maximum value of <upper bound> is implementation-defined (IL018). <upper bound>, if specified, shall not be greater than this value.
- 2) Every <graph pattern quantifier> is normalized, as follows:
 - a) <asterisk> is equivalent to:
 $\{0\}$
 - b) <plus sign> is equivalent to:
 $\{1\}$
 - c) If <fixed quantifier> FQ is specified, then let UI be the <unsigned integer> contained in FQ . FQ is equivalent to:
 $\{UI, UI\}$
 - d) If <general quantifier> GQ is specified, and if <lower bound> is not specified, then the <unsigned integer> 0 (zero) is supplied as the <lower bound>.
- 3) If <general quantifier> GQ is specified or implied by the preceding normalizations, then:

Case:

 - a) If <upper bound> is specified, then:
 - i) The value of <upper bound> VUP shall be greater than 0 (zero).
 - ii) The value of <lower bound> LUP shall be less than or equal to VUP .

- iii) If LUP equals VUP , then GQ is a *fixed quantifier*.
 - iv) GQ is a *bounded quantifier*.
 - b) Otherwise, GQ is an *unbounded quantifier*.
- 4) A <graph pattern quantifier> that is not a fixed quantifier is a *variable quantifier*.

General Rules

None.

Conformance Rules

« WG3:BER-030 »

- 1) Without Feature G060, “Bounded graph pattern quantifiers”, conforming SQL language shall not contain a <fixed quantifier> or a <general quantifier> that immediately contains an <upper bound>.
- 2) Without Feature G061, “Unbounded graph pattern quantifiers”, conforming SQL language shall not contain a <graph pattern quantifier> that immediately contains <asterisk>, <plus sign>, or a <general quantifier> that does not immediately contain an <upper bound>.

16.11 <simplified path pattern expression>

Function

Express a path pattern as a regular expression of edge labels.

Format

```

<simplified path pattern expression> ::=

  <simplified defaulting left>
  | <simplified defaulting undirected>
  | <simplified defaulting right>
  | <simplified defaulting left or undirected>
  | <simplified defaulting undirected or right>
  | <simplified defaulting left or right>
  | <simplified defaulting any direction>

<simplified defaulting left> ::=
  <left minus slash> <simplified contents> <slash minus>

<simplified defaulting undirected> ::=
  <tilde slash> <simplified contents> <slash tilde>

<simplified defaulting right> ::=
  <minus slash> <simplified contents> <slash minus right>

<simplified defaulting left or undirected> ::=
  <left tilde slash> <simplified contents> <slash tilde>

<simplified defaulting undirected or right> ::=
  <tilde slash> <simplified contents> <slash tilde right>

<simplified defaulting left or right> ::=
  <left minus slash> <simplified contents> <slash minus right>

<simplified defaulting any direction> ::=
  <minus slash> <simplified contents> <slash minus>

<simplified contents> ::=
  <simplified term>
  | <simplified path union>
  | <simplified multiset alternation>

<simplified path union> ::=
  <simplified term> <vertical bar> <simplified term>
  [ { <vertical bar> <simplified term> }... ]

<simplified multiset alternation> ::=
  <simplified term> <multiset alternation operator> <simplified term> [ { <multiset
  alternation operator> <simplified term> }... ]

<simplified term> ::=
  <simplified factor low>
  | <simplified concatenation>

<simplified concatenation> ::=
  <simplified term> <simplified factor low>

<simplified factor low> ::=
  <simplified factor high>
  | <simplified conjunction>

```

16.11 <simplified path pattern expression>

```

<simplified conjunction> ::==
  <simplified factor low> & <simplified factor high>

<simplified factor high> ::=
  <simplified tertiary>
  | <simplified quantified>
  | <simplified questioned>

<simplified quantified> ::=
  <simplified tertiary> <graph pattern quantifier>

<simplified questioned> ::=
  <simplified tertiary> ?<question mark>

<simplified tertiary> ::=
  <simplified direction override>
  | <simplified secondary>

<simplified direction override> ::=
  <simplified override left>
  | <simplified override undirected>
  | <simplified override right>
  | <simplified override left or undirected>
  | <simplified override undirected or right>
  | <simplified override left or right>
  | <simplified override any direction>

<simplified override left> ::=
  <left angle bracket> <simplified secondary>

<simplified override undirected> ::=
  <tilde> <simplified secondary>

<simplified override right> ::=
  <simplified secondary> <right angle bracket>

<simplified override left or undirected> ::=
  <left arrow tilde> <simplified secondary>

<simplified override undirected or right> ::=
  <tilde> <simplified secondary> <right angle bracket>

<simplified override left or right> ::=
  <left angle bracket> <simplified secondary> <right angle bracket>

<simplified override any direction> ::=
  <minus sign> <simplified secondary>

<simplified secondary> ::=
  <simplified primary>
  | <simplified negation>

<simplified negation> ::=
  !<exclamation mark> <simplified primary>

<simplified primary> ::=
  <label name>
  | <left paren> <simplified contents> <right paren>

```

** Editor's Note (number 168) **

It has been proposed that a macro name may be a <simplified primary> in a <simplified path pattern expression>. See [Language Opportunity \[GQL-034\]](#).

Syntax Rules

- 1) A <simplified negation> shall not contain a <simplified concatenation>, <simplified quantified>, <simplified questioned>, or <simplified multiset alternation>.
- 2) A <simplified direction override> shall not contain another <simplified direction override>.
- 3) A <simplified direction override> shall not contain <simplified concatenation>, <simplified quantified>, <simplified questioned>, or <simplified multiset alternation>.
- 4) A <simplified conjunction> shall not contain a <simplified concatenation>, <simplified quantified>, <simplified questioned>, or <simplified multiset alternation>.
- 5) A <simplified path pattern expression> *SPPE* is replaced by:

(*SPPE*)

NOTE 130 — This is done once for each <simplified path pattern expression> prior to the following recursive transformation and not with each iteration of the transformation.

- 6) The following rules are recursively applied until no <simplified path pattern expression>s remain.

NOTE 131 — The rules work from the root of the parse tree of a <simplified path pattern expression>. At each step, the coarsest analysis of a <simplified path pattern expression> is replaced, eliminating at least one level of the parse tree, measured from the root. Note that each replacement may create more <simplified path pattern expression>s than before, but these replacements have less depth. Eventually the recursion replaces <simplified path pattern expression> with <edge pattern>.

- a) Let *SPPE* be a <simplified path pattern expression>.
 - i) Let *SC* be the <simplified contents> contained in *SPPE*.
 - ii) Let *PREFIX* be the <minus slash>, <left minus slash>, <tilde slash>, <left tilde slash>, or <left minus slash> contained in *SPPE*.
 - iii) Let *SUFFIX* be the <slash minus right>, <slash minus>, <slash tilde>, or <slash tilde right> contained in *SPPE*.
 - iv) Let *EDGEPRE* and *EDGESUF* be determined by [Table 2, “Conversion of simplified syntax delimiters to default edge delimiters”](#), from the row containing the values of *PREFIX* and *SUFFIX*.

Table 2 — Conversion of simplified syntax delimiters to default edge delimiters

<i>PREFIX</i>	<i>SUFFIX</i>	<i>EDGEPRE</i>	<i>EDGESUF</i>
-/	/->	-[]->
<-/	/-	<-[]-
~/	/~	~[]~
~/	/~>	~[]~>
<~/	/~	<~[]~
<-/	/->	<-[]->
-/	/-	-[]-

b) Case:

« WG3:BER-080 »

- i) If SC is a <simplified path union> SPU , then let N be the number of <simplified term>s simply contained in SPU , and let ST_1, \dots, ST_N be those <simplified term>s; $SPPE$ is replaced by:

$PREFIX\ ST_1\ SUFFIX\ | PREFIX\ ST_2\ SUFFIX\ | \dots\ | PREFIX\ ST_N\ SUFFIX$

« WG3:BER-080 »

- ii) If SC is a <simplified multiset alternation> SMA , then let N be the number of <simplified term>s simply contained in SMA , and let ST_1, \dots, ST_N be those <simplified term>s; $SPPE$ is replaced by:

$PREFIX\ ST_1\ SUFFIX\ |+| PREFIX\ ST_2\ SUFFIX\ |+| \dots\ |+| PREFIX\ ST_N\ SUFFIX$

« WG3:BER-080 »

- iii) If SC is a <simplified concatenation> $SCAT$, then let ST be the <simplified term> and let SFL be the <simplified factor low> simply contained in $SCAT$; $SPPE$ is replaced by:

$PREFIX\ ST\ SUFFIX\ PREFIX\ SFL\ SUFFIX$

- iv) If SC is a <simplified conjunction> $SAND$, then $SPPE$ is replaced by:

$EDGEPRE\ IS\ SAND\ EDGESUF$

NOTE 132 — As a result, $SAND$ is now interpreted as a <label expression> within an <edge pattern>. By earlier Syntax Rules, there are no operators allowed in $SAND$ that cannot be interpreted as operators of a <label expression>.

« WG3:BER-080 »

- v) If SC is a <simplified quantified> SQ , then let ST be the <simplified tertiary> simply contained in SC and let GPQ be the <graph pattern quantifier> simply contained in SQ ; $SPPE$ is replaced by:

$(\ PREFIX\ ST\ SUFFIX\)\ GPQ$

« WG3:BER-080 »

- vi) If SC is a <simplified questioned> SQU , then let ST be the <simplified tertiary> simply contained in SC ; $SPPE$ is replaced by:

$(\ PREFIX\ ST\ SUFFIX\)\ ?$

- vii) If SC is a <simplified direction override> SDO , then let SS be the <simplified secondary> simply contained in SDO .

Case:

NOTE 133 — As a result of the following replacements, SDO is now interpreted as a <label expression> within an <edge pattern>. By earlier Syntax Rules, there are no operators allowed in SDO that cannot be interpreted as operators of a <label expression>.

- 1) If SDO is <simplified override left>, then $SPPE$ is replaced by:

$<-[\ IS\ SS\]-$

- 2) If SDO is <simplified override undirected>, then $SPPE$ is replaced by:

$\sim[\ IS\ SS\]\sim$

16.11 <simplified path pattern expression>

- 3) If *SDO* is <simplified override left or undirected>, then *SPPE* is replaced by:

`<~[IS SS]~`

- 4) If *SDO* is <simplified override undirected or right>, then *SPPE* is replaced by:

`~[IS SS]~>`

- 5) If *SDO* is <simplified override left or right>, then *SPPE* is replaced by:

`<- [IS SS]->`

- 6) If *SDO* is <simplified override any direction>, then *SPPE* is replaced by:

`-[IS SS]-`

- viii) If *SC* is a <simplified negation> *SN*, then *SPPE* is replaced by:

`EDGEPR IS SN EDGESUF`

NOTE 134 — As a result, *SN* is now interpreted as a <label expression> within an <edge pattern>. By earlier Syntax Rules, there are no operators allowed in *SN* that cannot be interpreted as operators of a <label expression>.

- ix) If *SC* is a <simplified primary> *SP*, then

Case:

- 1) If *SP* is a <label name>, then *SPPE* is replaced by:

`EDGEPR IS SP EDGESUF`

« WG3:BER-080 »

- 2) Otherwise, let *INNER* be the <simplified contents> simply contained in *SC*; *SPPE* is replaced by:

`(PREFIX INNER SUFFIX)`

« WG3:BER-030 »

- 7) The Conformance Rules of Subclause 16.7, “<path pattern expression>” are applied to the result of the previous syntactic transformation.

General Rules

None.

Conformance Rules

« WG3:BER-030 deleted two CRs. »

« WG3:BER-030 »

- 1) Without Feature G080, “Simplified Path Pattern Expression: basic defaulting”, conforming SQL language shall not contain a <simplified defaulting left>, a <simplified defaulting right>, or a <simplified defaulting any direction>.
- 2) Without Feature G081, “Simplified Path Pattern Expression: full overrides”, conforming SQL language shall not contain a <simplified direction override> that is not a <simplified override left>, <simplified override right>, or a <simplified override any direction>.

16.11 <simplified path pattern expression>

- 3) Without Feature G082, “Simplified Path Pattern Expression: basic overrides”, conforming SQL language shall not contain a <simplified override left>, a <simplified override right>, or a <simplified override any direction>.

16.12 <where clause>

Function

Compute a new binding table by selecting records from the current working table fulfilling the specified <search condition>.

Format

```
<where clause> ::=  
    WHERE <search condition>
```

Syntax Rules

« BER-019 »

- 1) Let WC be the <where clause> and let SC be the <search condition> immediately contained in WC .
- 2) The declared type of the incoming working record of SC is the declared type of the incoming working record of WC amended with the recording type of the declared type of the incoming working table of WC .
- 3) The declared type of the incoming working table of SC is the material unit binding table type.
- 4) The declared type of WC is the declared type of incoming working table of WC .

General Rules

« BER-019 One rule deleted »

- 1) Let $WHERE$ be a new empty binding table.
- 2) For each record R of the current working table:
 - a) Let $INCLUDE$ be the result of SC in a new child execution context amended with R .
 - b) If $INCLUDE$ is True, then add R to $WHERE$.
- 3) The result of the application of this Subclause is $WHERE$.

Conformance Rules

None.

16.13 <procedure call>

**** Editor's Note (number 169) ****

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:BNE-023], needs further discussion to establish consensus. See Possible Problem [GQL-118](#).

Function

Define a <procedure call>.

Format

```
<procedure call> ::=  
  <inline procedure call>  
  | <named procedure call>
```

**** Editor's Note (number 170) ****

Consider supporting grouped procedure calls. Regular procedure calls execute the specified procedure for each record of the incoming binding table and combine the binding tables returned by each such call into a final result. Grouped procedure calls execute the specified procedure for each partition of the incoming binding table using some specified grouping key. Attention would be required to ensure that every variable is correctly declared as fixed or iterated (binding) variable in each such procedure call depending on if it is computed from the grouping key, and constants only or not. See Language Opportunity [GQL-186](#).

Syntax Rules

- 1) Let PC be the <procedure call>.
- 2) Case:
 - a) If the <named procedure call> NPC is specified, then the declared type of PC is the declared type of NPC .
 - b) Otherwise, the <inline procedure call> IPC is specified, and the declared type of PC is the declared type of IPC .

General Rules

« WG3:BER-099R1 deleted two GRs by implication »

« WG3:BER-099R1 »

- 1) The outcome of the application of these General Rules is the current execution outcome.

Conformance Rules

None.

16.14 <inline procedure call>

** Editor's Note (number 171) **

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:BNF-023], needs further discussion to establish consensus. See Possible Problem [GQL-119](#).

Function

Define an <inline procedure call>.

Format

```
<inline procedure call> ::=  
  <nested procedure specification>
```

Syntax Rules

** Editor's Note (number 172) **

BER-019 has established declared type propagation for working record, working table, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-271](#).

- 1) Let *IPC* be the <inline procedure call>, and let *PROC* be the <nested procedure specification> that is immediately contained in *IPC*.
- 2) Let *PROCSIG* be the procedure signature of *PROC*.

** Editor's Note (number 173) **

Rules for the derivation of the procedure signature from a <nested procedure specification>, including the correct handling of <parameter cardinality> need to be specified. See Possible Problem [GQL-021](#).

- 3) The declared type of *IPC* is the expected result type of *PROCSIG*.

General Rules

- 1) Let *CONTEXT* be the current execution context.
- 2) The following steps are performed in a new child execution context with the working table of *CONTEXT* as its working table:
 - a) Execute *PROC* with no arguments.

** Editor's Note (number 174) **

Further details need to be provided. See Possible Problem [GQL-119](#).

« WG3:BER-099R1 »

- b) The outcome of the application of these General Rules is the current execution outcome.

Conformance Rules

None.

16.15 <named procedure call>

**** Editor's Note (number 175) ****

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:BNF-023], needs further discussion to establish consensus. See Possible Problem [GQL-120](#).

**** Editor's Note (number 176) ****

Decide on support for passing procedures, queries, functions as parameters. See Language Opportunity [GQL-023](#).

**** Editor's Note (number 177) ****

Bindings for host languages should eventually be defined. See Language Opportunity [GQL-003](#).

Function

Define a <named procedure call>.

Format

```

<named procedure call> ::= 
  <procedure reference> <left paren> [ <procedure argument list> ] <right paren>
  [ <yield clause> ]

<procedure argument list> ::= 
  <procedure argument> [ { <comma> <procedure argument> }... ]

<procedure argument> ::= 
  <value expression>

```

Syntax Rules

**** Editor's Note (number 178) ****

BER-019 has established declared type propagation for working record, working table, and result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-272](#).

**** Editor's Note (number 180) ****

Handling of default values for optional parameters needs to be specified. See Possible Problem [GQL-120](#).

- 1) Let *NPC* be the <named procedure call> and let *PROC* be the procedure referenced by the <procedure reference> that is immediately contained in *NPC*.

**** Editor's Note (number 179) ****

This needs to be detailed further; in particular it is necessary to describe how *PROC* is to be resolved statically (to determine its signature) vs. dynamically (to execute it). This may require re-determining which Rules in this Subclause are SRs and which are GRs. See Possible Problem [GQL-120](#).

- 2) Let *PROCSIG* be the procedure signature of *PROC*.
- 3) Let *ARGEEXPS* be the sequence of all <value expression>s that are simply contained in *NPC* in the order of their occurrence in *NPC* from left to right and let *NUMARGS* be the number of such elements in *ARGEEXPS*.

- 4) If $NUMARGS$ is less than the number of arguments required by $PROCSIG$, then an exception condition is raised: *procedure call error — incorrect number of arguments (G0001)*.
- 5) If $NUMARGS$ is greater than the number of arguments allowed by $PROCSIG$, then an exception condition is raised: *procedure call error — incorrect number of arguments (G0001)*.
- 6) Let $ARGEXP_i$, $1 \leq i \leq NUMARGS$, be the i -th element of $ARGEXPS$.
- 7) If there is an $ARGEXP_i$, $1 \leq i \leq NUMARGS$, whose declared type is known but which is not a subtype of the required procedure parameter type of the i -th procedure parameter of $PROCSIG$, then an exception condition is raised: *procedure call error — invalid parameter type (G0002)*.

**** Editor's Note (number 181) ****

Furthermore, handling of <parameter cardinality> needs to be specified. See Possible Problem [GQL-120](#).

- 8) If NPC immediately contains a <yield clause> and the expected result type of $PROCSIG$ is not a binding table type, then an exception condition is raised: *procedure call error — invalid result type (G0003)*.
- 9) The declared type of NPC is the expected result type of $PROCSIG$.

General Rules

- 1) Let $ARGV_i$ be the result of $ARGEXP_i$, $1 \leq i \leq NARGS$.
- 2) If there is an $ARGV_i$, $1 \leq i \leq NUMARGS$, such that $ARGV_i$ is not of the required procedure parameter type of the i -th procedure parameter of $PROCSIG$, then an exception condition is raised: *procedure call error — invalid parameter type (G0002)*.
- 3) Let $CONTEXT$ be the current execution context.
- 4) Let R be a new record comprising fields F_i , $1 \leq i \leq NUMARGS$, such that the field name of F_i is the procedure parameter name of the i -th procedure parameter of $PROCSIG$ and the field value of F_i is $ARGV_i$.
- 5) The following steps are performed in a new child execution context with R as its working record:

- a) Execute $PROC$.

**** Editor's Note (number 182) ****

Further details need to be provided. See Possible Problem [GQL-120](#).

- b) If a <yield clause> YC is specified, then:

- i) If the current execution outcome is a successful outcome with a binding table result, then set the current working table to the result of the current execution outcome; otherwise, an exception condition is raised:

**** Editor's Note (number 183) ****

Exception condition to be determined. See Possible Problem [GQL-158](#).

- ii) The General Rules of YC are applied; let $YIELD$ be the $YIELD$ returned from the application of these General Rules.

iii) Set the current execution outcome to a successful outcome with *YIELD* as its result.
« WG3:BER-099R1 »

c) The outcome of the application of these General Rules is the current execution outcome.

Conformance Rules

None.

16.16 <yield clause>

Function

Restructure a binding table.

Format

```

<yield clause> ::=

    YIELD <yield item list>

<yield item list> ::=

    <yield item> [ { <comma> <yield item> }... ]

<yield item> ::=

    { <yield item name> [ <yield item alias> ] }

<yield item name> ::=

    <identifier>

<yield item alias> ::=

    AS <variable name>
  
```

Syntax Rules

- 1) Let YC be the <yield clause>.
- 2) Let YIL be the collection of <yield item>s simply contained in YC .
- 3) Let n be the number of <yield item>s in YIL .
- 4) For each <yield item> YI_i , $1 \leq i \leq n$, from YIL :
 - a) Let YIN_i be the <identifier> immediately contained in the <yield item name> specified by YI_i .
 - b) If YI_i does not immediately contain a <yield item alias>, then:
 - i) YIN_i shall be a <variable name>.
 - ii) YI_i is effectively replaced by:

YIN_i AS YIN_i

« BER-019 »

- 5) The declared type of YC is a binding table type BTT defined as follows:
 - a) BTT has n columns.
 - b) For each <yield item> YI_i from YIL , $1 \leq i \leq n$:
 - i) Let YIA_i be the <variable name> contained in the <yield item alias> specified by YI_i .
 - ii) Let YIN_i be the <yield item name> specified by YI_i .
 - iii) The declared type of the incoming working table shall have a column SC with column name YIN_i .
 - iv) BTT has a column TC .

- v) The column name of TC is YIA_i .
- vi) The column type of TC is the column type of the SC .

General Rules

- 1) Let $YIELD$ be a new empty binding table.
- 2) For each record R of the current working table in a new child execution context amended with R :
 - a) Let T be a new empty record.
 - b) For each <yield item> YI_i , $1 \leq i \leq n$, from YIL :
 - i) Let YIN_i be the <yield item name> specified by YI_i .
 - ii) Let F_i be the field of the current working record whose field name is YIN_i .
 - iii) Let YIV_i be the field value of F_i .

NOTE 135 — As opposed to the General Rules for <binding variable>, <yield item>s only consider the current working record and ignore the working records of any parent execution contexts of the current execution context in the current execution stack.
 - iv) Let YIA_i be the <variable name> contained in the <yield item alias> specified by YI_i .
 - v) Add a new field with field name YIA_i and with field value YIV_i to T .
 - c) Add T to $YIELD$.
- 3) The result of the application of this Subclause is $YIELD$.

16.17 <group by clause>

** Editor's Note (number 184) **

WG3:W05-020 suggests that discussion of this Subclause is needed to establish consensus. See Possible Problem [GQL-078](#).

Function

Define a <group by clause> for specifying the set of grouping keys to be used during grouping.

** Editor's Note (number 185) **

Aggregation functionality should be improved for the needs of GQL. See Language Opportunity [GQL-017](#).

Format

```
<group by clause> ::=  
    GROUP BY <grouping element list>  
  
<grouping element list> ::=  
    <grouping element> [ { <comma> <grouping element> } ]  
    | <empty grouping set>  
  
<grouping element> ::=  
    <binding variable>  
  
<empty grouping set> ::=  
    <left paren> <right paren>
```

Syntax Rules

** Editor's Note (number 186) **

BER-019 has established declared type propagation for working record, working table, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-273](#).

- 1) Every <binding variable> shall unambiguously reference a column of the current working table.

General Rules

- 1) Let *GBC* be the <group by clause> and let *GEL* be the <grouping element list>.

- 2) Case:

« BER-019 - email from Hannes Voigt 2022-06-22 »

- a) If *GEL* is the <empty grouping set>, then let *GROUP_BY* be a unit binding table.
- b) Otherwise:
 - i) Let *GESEQ* be the sequence of <grouping element>s immediately contained in *GEL* and let *NGESEQ* the number of such <grouping element>s in *GESEQ*.
 - ii) Let *GE_i*, $1 \leq i \leq NGESEQ$, be *i*-th element of *GESEQ* and let *GEBV_i* be the <binding variable> simply contained in *GEBV_i*.

- iii) Let $GROUP_BY$ be a new empty duplicate-free binding table.
- iv) For each record R of the current working table in a new child execution context amended with R :
 - 1) Let T be a new record comprising fields F_i , $1 \leq i \leq NGESEQ$, such that the field name of F_i is $GEBV_i$ and the field value of F_i is the value of $GEBV_i$.
 - 2) If $GROUP_BY$ does not contain T , then T is added.
- 3) The result of the application of this Subclause is $GROUP_BY$.

Conformance Rules

None.

16.18 <order by clause>

Function

Define an <order by clause> for obtaining an ordered binding table from the current working table.

Format

```
<order by clause> ::=  
    ORDER BY <sort specification list>
```

Syntax Rules

None.

General Rules

- 1) Let *OBC* be the <order by clause>. Let *SSL* be the <sort specification list> immediately contained in *OBC*.
- 2) Let *ORDER_BY* be a new ordered binding table created from a collection of all records of the current working table sorted by applying the General Rules of Subclause 16.20, “<sort specification list>”, to *SSL*.
- 3) The result of the application of this Subclause is *ORDER_BY*.

Conformance Rules

None.

16.19 <aggregate function>

**** Editor's Note (number 187) ****

WG3:W05-020 suggests that this Subclause, which was discussed in [WG3:BNE-023], needs further discussion to establish consensus. See Possible Problem [GQL-123](#).

**** Editor's Note (number 188) ****

Aggregation functionality should be improved for the needs of GQL. See Language Opportunity [GQL-017](#).

**** Editor's Note (number 189) ****

It is strongly suspected that the Syntax and General Rules of this Subclause are incomplete. These rules must be reviewed and completed as necessary. See Possible Problem [GQL-205](#).

Function

Specify a value computed from a collection of rows.

Format

```
<aggregate function> ::=  
    COUNT <left paren> <asterisk> <right paren>  
    | <general set function>  
    | <binary set function>
```

**** Editor's Note (number 190) ****

Consider inclusion of aggregate function calls to procedures with formal parameters of multiple parameter cardinality. See Language Opportunity [GQL-186](#).

```
<general set function> ::=  
    <general set function type>  
        <left paren> <set quantifier> <value expression> <right paren>  
  
<binary set function> ::=  
    <binary set function type>  
        <left paren> <dependent value expression> <comma> <independent value expression>  
        <right paren>  
  
<general set function type> ::=  
    AVG  
    | COUNT  
    | MAX  
    | MIN  
    | SUM  
    | PRODUCT  
    | COLLECT  
    | stDev  
    | stDevP  
  
<set quantifier> ::=  
    DISTINCT  
    | ALL  
  
<binary set function type> ::=  
    percentileCont  
    | percentileDist
```

```
<dependent value expression> ::=  
  [ <set quantifier> ] <numeric value expression>  
  
<independent value expression> ::=  
  <numeric value expression>
```

Syntax Rules

**** Editor's Note (number 191) ****

BER-019 has established declared type propagation for working record, working table, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-274](#).

- 1) Let *AF* be the <aggregate function>.
- 2) If <general set function> is specified and <set quantifier> is not specified, then ALL is implicit.
- 3) If <dependent value expression> is specified and <set quantifier> is not specified, then ALL is implicit.
- 4) *AF* shall not contain a <procedure body>.

**** Editor's Note (number 192) ****

This could be relaxed but would require careful adjustment of the scoping rules in <return statement>. See Possible Problem [GQL-172](#).

- 5) A <general set function> shall not contain an <aggregate function>.
- 6) A <dependent value expression> shall not contain an <aggregate function>.
- 7) If COUNT is specified, then the declared type of the result is an implementation-defined ([ID059](#)) exact numeric type with scale 0 (zero).

General Rules

- 1) Let *TABLE* be the current working table.
- 2) Case:
 - a) If *AF* immediately contains the <general set function> *GSF*, then let *EXP* be the <value expression> immediately contained in *GSF* and let *SQ* be the <set quantifier> immediately contained in *GSF*.
 - b) Otherwise, *AF* immediately contains the <binary set function> *BSF*. Let *EXP* be the <dependent value expression> immediately contained in *BSF* and let *SQ* be the <set quantifier> immediately contained in *EXP*.
- 3) Let *VALUES* be a new empty collection.
- 4) For each record *R* of *TABLE* in a new child execution context amended with *R*:
 - a) Let *EXPRE* be the result of *EXP*.
 - b) If *SQ* is DISTINCT and *EXPRE* is not in *VALUES*, then add *EXPRE* to *VALUES*; otherwise, add *EXPRE* to *VALUES*.
- 5) Case:

- a) If AF immediately contains the <general set function> GSF , then let $RESULT$ be the result of evaluating GSF on $VALUES$.
- b) Otherwise, AF immediately contains the <binary set function> BSF . Let IVE be the <independent value expression> immediately contained in BSF , let $IVERE$ be the result of evaluating IVE , and let $RESULT$ be the result of evaluating BSF on $VALUES$ and $IVERE$.

**** Editor's Note (number 193) ****

Details regarding the computation of supported aggregate functions need to be specified. See [Language Opportunity GQL-017](#).

- 6) The result of evaluating AF is $RESULT$.

Conformance Rules

None.

16.20 <sort specification list>

Function

Specify a sort order.

Format

```

<sort specification list> ::= 
  <sort specification> [ { <comma> <sort specification> }... ]

<sort specification> ::= 
  <sort key> [ <ordering specification> ] [ <null ordering> ]

<sort key> ::= 
  <value expression>

<ordering specification> ::= 
  ASC
  | DESC

<null ordering> ::= 
  NULLS FIRST
  | NULLS LAST

```

Syntax Rules

**** Editor's Note (number 194) ****

BER-019 has established declared type propagation for working record, working table, and result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-275](#).

- 1) Each <value expression> immediately contained in the <sort key> contained in a <sort specification> is an operand of an ordering operation. The Syntax Rules and Conformance Rules of Subclause 21.9, "Ordering operations", apply.
- 2) Let *SSL* be the <sort specification list>
- 3) Let *NSS* be the number of <sort specification>s immediately contained in *SSL*.
- 4) Let SS_i , $1 \leq i \leq NSS$, be the *i*-th <sort specification> immediately contained in *SSL*.
- 5) For each SS_i , $1 \leq i \leq NSS$:
 - a) Let SK_i be the <sort key> immediately contained in SS_i .
 - b) If SS_i does not immediately contain an <ordering specification>, then SS_i is effectively replaced by:
 SK_i ASC
- 6) If <null ordering> is not specified, then an implementation-defined ([ID060](#)) <null ordering> is implicit. The implementation-defined default for <null ordering> shall not depend on the context outside of <sort specification list>.

General Rules

- 1) A <sort specification list> defines an ordering of rows, as follows:
 - a) Let N be the number of <sort specification>s.
 - b) Let K_i , $1 \leq i \leq N$, be the <sort key> contained in the i -th <sort specification>.
 - c) Each <sort specification> specifies the *sort direction* for the corresponding sort key K_i . If DESC is not specified in the i -th <sort specification>, then the sort direction for K_i is ascending and the applicable <comp op> is the <less than operator>; otherwise, the sort direction for K_i is descending and the applicable <comp op> is the <greater than operator>.
 - d) Let P be any record of the collection of rows to be ordered, and let Q be any other record of the same collection of rows.
 - e) Let PV_i and QV_i be the values of K_i in P and Q , respectively. The relative position of rows P and Q in the result is determined by comparing PV_i and QV_i as follows:
 - i) The comparison is performed according to the General Rules of Case:
 - 1) If the declared type of K_i is a record type, then this Subclause, applied recursively.
 - 2) Otherwise, Subclause 18.3, “<comparison predicate>”, where the <comp op> is the applicable <comp op> for K_i .
 - ii) The comparison is performed with the following special treatment of null values. Case:
 - 1) If PV_i and QV_i are both the null value, then they are considered equal to each other.
 - 2) If PV_i is the null value and QV_i is not the null value, then
 - Case:
 - A) If NULLS FIRST is specified or implied, then $PV_i <\text{comp op}> QV_i$ is considered to be True.
 - B) If NULLS LAST is specified or implied, then $PV_i <\text{comp op}> QV_i$ is considered to be False.
 - 3) If PV_i is not the null value and QV_i is the null value, then
 - Case:
 - A) If NULLS FIRST is specified or implied, then $PV_i <\text{comp op}> QV_i$ is considered to be False.
 - B) If NULLS LAST is specified or implied, then $PV_i <\text{comp op}> QV_i$ is considered to be True.
 - f) PV_i is said to *precede* QV_i if the result of the <comparison predicate> “ $PV_i <\text{comp op}> QV_i$ ” is True for the applicable <comp op>.
 - g) If PV_i and QV_i are not the null value and the result of “ $PV_i <\text{comp op}> QV_i$ ” is Unknown, then the relative ordering of PV_i and QV_i is implementation-dependent (US007).

- h) The relative position of record P is before record Q if PV_n precedes QV_n for some n , $1 \leq n \leq N$, and PV_i is not distinct from QV_i for all $i < n$.
- i) Two rows that are not distinct with respect to the <sort specification>s are said to be *peers* of each other. The relative ordering of peers is implementation-dependent ([US006](#)).

Conformance Rules

None.

16.21 <limit clause>

Function

Define a <limit clause> for obtaining a new binding table that retains only a limited number of records of the current working table.

Format

```
<limit clause> ::=  
    LIMIT <unsigned integer specification>
```

** Editor's Note (number 195) **

WITH TIES, ONLY, RECORDS, and GROUPS to be added. See Language Opportunity [GQL-161](#)

Syntax Rules

None.

General Rules

- 1) Let *TABLE* be the current working table.
- 2) If *TABLE* is not ordered, then let *ORDERED_TABLE* be a new ordered binding table created from the result of sorting the collection of all records of *TABLE* according to an implementation-dependent (US001) order; otherwise, let *ORDERED_TABLE* be *TABLE*.
- 3) Let *LIMIT* be a new ordered binding table obtained by selecting only the first *V* records of *ORDERED_TABLE* and discarding all subsequent records.
- 4) The result of the application of this Subclause is *LIMIT*.

Conformance Rules

None.

16.22 <offset clause>

Function

Define an <offset clause> for obtaining a new binding table that retains all records of the current working table except for some discarded initial records.

Format

```
<offset clause> ::=  
  <offset synonym> <unsigned integer specification>  
  
<offset synonym> ::=  
  OFFSET | SKIP
```

** Editor's Note (number 196) **

WITH TIES, ONLY, RECORDS, and GROUPS to be added. See Language Opportunity [GQL-162](#).

Syntax Rules

None.

General Rules

- 1) Let *TABLE* be the current working table.
- 2) If *TABLE* is not ordered, then let *ORDERED_TABLE* be a new ordered binding table created from the result of sorting the collection of all records of *TABLE* according to an implementation-dependent (US001) order; otherwise, let *ORDERED_TABLE* be *TABLE*.
- 3) Let *OFFSET* be a new ordered binding table obtained from all but the first *V* records of *ORDERED_TABLE*.
- 4) The result of the application of this Subclause is *OFFSET*.

Conformance Rules

None.

17 Object references

** Editor's Note (number 197) **

WG3:W05-020 suggests that this clause, which is based on [WG3:MMX-055] and discussed in [WG3:W03-034], needs further discussion to establish consensus. See Possible Problem [GQL-127](#).

17.1 Schema references

Function

Define a schema reference.

Format

```
<schema reference> ::=  
  <predefined schema parameter>  
  | <catalog schema parent and name>  
  | <external object reference>  
  
<catalog schema parent and name> ::=  
  [ <absolute url path> ] <solidus> <schema name>  
  | <curl path parameter>
```

Syntax Rules

« WG3:BER-088R1 »

- 1) If *<schema reference> SR* is specified, then the GQL-schema identified by *SR* is the GQL-schema identified by the immediately contained *<predefined schema parameter>*, *<catalog schema parent and name>*, or *<external object reference>*.
- 2) If the *<catalog schema parent and name> CSPN* is specified, then:

a) After applying all relevant syntactic transformations, let *PARENT* be determined as follows

Case:

« WG3:BER-088R1 »

- i) If *CSPN* does not immediately contain an *<absolute url path>*, then *PARENT* is the catalog root.
 - ii) Otherwise, *PARENT* is the catalog object identified by the immediately contained *<absolute url path>*.
 - iii) *PARENT* shall be a GQL-directory.
- b) The schema identified by *CSPN* is defined as follows:
 - i) Let *SN* be the *<schema name>* immediately contained in *CSPN*.
 - ii) *SN* shall identify an existing schema descriptor in *PARENT*.

- | iii) The schema identified by *CSPN* is the GQL-schema identified by *SN* in *PARENT*.

General Rules

None.

Conformance Rules

None.

17.2 Graph references

Function

Specify a graph reference.

Format

```

<graph reference> ::= 
    <graph resolution expression>
  | <local graph reference>

<graph resolution expression> ::= 
    [ PROPERTY ] GRAPH <catalog graph reference>

<catalog graph reference> ::= 
    <catalog graph parent and name>
  | <predefined graph parameter>
  | <external object reference>

<catalog graph parent and name> ::= 
    <graph parent specification> <graph name>
  | <curl path parameter>

<graph parent specification> ::= 
    [ <parent catalog object reference> ] [ <qualified object name> <period> ]

<local graph reference> ::= 
    <qualified graph name>

<qualified graph name> ::= 
    [ <qualified object name> <period> ] <graph name>
  
```

Syntax Rules

**** Editor's Note (number 198) ****

BER-019 has established declared type propagation for working record, working table, and result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-276](#).

- 1) If the <graph parent specification> *GPS* does not immediately contain a <parent catalog object reference>, then a <parent catalog object reference> of . / is implicit.

General Rules

- 1) If the <graph reference> *GR* is specified, then the result of *GR* is the result of the immediately contained <graph resolution expression> or <local graph reference>.
- 2) If the <graph resolution expression> *GRE* is specified, then the result of *GRE* is the result of the immediately contained <catalog graph reference>.
- 3) If the <catalog graph reference> *CGR* is specified, then:
 - a) Let *ROCGR* be determined as follows

Case:

- i) If *CGR* simply contains the <graph parent specification> *GPS* and the <graph name> *GN*, then *ROCGR* is the result of resolving *GN* with the sequence comprising only the result of *GPS* as the start object candidate sequence, as specified by [General Rule 7](#).
 - ii) If *CGR* immediately contains the <predefined graph parameter> *PGP*, then *ROCGR* is the result of *PGP*.
 - iii) If *CGR* immediately contains an <external object reference> *EOR*, then the result of *ROCGR* is the result of *EOR*.
- b) If *ROCGR* is a graph, then the result of *CGR* is *ROCGR*; otherwise, an exception condition is raised:

** Editor's Note (number 199) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

- 4) If the <graph parent specification> *GPS* is specified, then let *ROPCOR* be the result of the <parent catalog object reference> immediately contained in *GPS* and define the result of *GPS* as follows
- Case:
- « WG3:BER-082R1 »
- a) If *GPS* immediately contains the <qualified object name> *QON*, then the result of *GPS* is the result of resolving *QON* with the sequence comprising only *ROPCOR* as the start object candidate sequence, as specified by [General Rule 3](#) of Subclause 17.8, “<qualified object name>”.
 - b) Otherwise, the result of *GPS* is *ROPCOR*.
- 5) If the <local graph reference> *LGR* is specified, then the result of *LGR* is the result of resolving the <qualified graph name> immediately contained in *LGR* with the current working record with appended current working schema as the start object candidate sequence, as specified by [General Rule 6](#).
- 6) If the <qualified graph name> *QGN* is specified, then *QGN* is resolved with the start object candidate sequence *SOCSEQ* as follows:

« WG3:BER-082R1 »

- a) If *QGN* immediately contains the <qualified object name> *QON*, then *SOCSEQ'* is the sequence comprising the result of resolving *QON* with *SOCSEQ* as the start object candidate sequence, as specified by [General Rule 3](#) of Subclause 17.8, “<qualified object name>”; otherwise, *SOCSEQ'* is *SOCSEQ*.
 - b) Let *GN* be the <graph name> immediately contained in *QGN* and let *RES* be the result of resolving *GN* with *SOCSEQ'* as the start object candidate sequence, as specified by [General Rule 7](#).
 - c) The result of resolving *QGN* with *SOCSEQ* as the candidate start object sequence is *RES*.
- 7) If the <graph name> *GN* is specified, then *GN* is resolved with the start object candidate sequence *SOCSEQ* as follows:
- a) Let *NSOCSEQ* be the number of elements of *SOCSEQ*, let *SOC_i*, $1 \leq i \leq NSOCSEQ$, be the *i*-th element of *SOCSEQ*, and let *RES_i* be determined as follows

Case:

- i) If SOC_i is a GQL-object with named components that contains a graph G with name GN , then RES_i is G .
 - ii) If SOC_i is a GQL-object with named components that contains an alias with name GN that resolves to a graph G , then RES_i is G .
 - iii) If SOC_i is a GQL-object with named components that contains a simple view SV with name GN that returns a graph, then RES_i is the graph obtained by calling SV with no arguments.
 - iv) If SOC_i is a GQL-object with named components that contains an alias with name GN that resolves to a simple view SV that returns a graph, then RES_i is the graph obtained by calling SV with no arguments.
 - v) Otherwise, RES_i is the null value.
- b) If there is a k , $1 \leq k \leq NSOCSEQ$, such that RES_k is not the null value and such that for every j , $1 \leq j < k$, it holds that RES_j is the null value, then the result of resolving GN with the start object candidate sequence $SOCSEQ$ is RES_k ; otherwise, an exception condition is raised:

** Editor's Note (number 200) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

Conformance Rules

None.

17.3 Graph type references

Function

Specify a graph type reference.

Format

```

<graph type reference> ::==
  <graph type resolution expression>
  | <local graph type reference>

<graph type resolution expression> ::==
  [ PROPERTY ] GRAPH TYPE <catalog graph type reference>

<catalog graph type reference> ::==
  <catalog graph type parent and name>
  | <external object reference>

<catalog graph type parent and name> ::==
  <graph type parent specification> <graph type name>
  | <curl path parameter>

<graph type parent specification> ::==
  [ <parent catalog object reference> ] [ <qualified object name> <period> ]

<local graph type reference> ::==
  <qualified graph type name>

<qualified graph type name> ::==
  [ <qualified object name> <period> ] <graph type name>

```

Syntax Rules

- 1) If the <graph type parent specification> *GTPS* does not immediately contain a <parent catalog object reference>, then a <parent catalog object reference> of ./ is implicit.

General Rules

- 1) If the <graph type reference> *GTR* is specified, then the result of *GTR* is the result of the immediately contained <graph type resolution expression> or <local graph type reference>.
- 2) If the <graph type resolution expression> *GTRE* is specified, then the result of *GTRE* is the result of the immediately contained <catalog graph type reference>.
- 3) If the <catalog graph type reference> *CGTR* is specified, then:
 - a) Let *ROCGTR* be determined as follows

Case:

 - i) If *CGTR* simply contains the <graph type parent specification> *GTPS* and the <graph type name> *GTN*, then *ROCGTR* is the result of resolving *GTN* with the sequence comprising only the result of *GTPS* as the start object candidate sequence, as specified by [General Rule 7](#).
 - ii) If *CGTR* immediately contains an <external object reference> *EOR*, then the result of *ROCGTR* is the result of *EOR*.

17.3 Graph type references

- b) If $ROCGTR$ is a graph type, then the result of $CGTR$ is $ROCGTR$; otherwise, an exception condition is raised:

** Editor's Note (number 201) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

« WG3:BER-082R1 »

- 4) If the <graph type parent specification> $GTPS$ is specified, then let $ROPCOR$ be the result of the <parent catalog object reference> immediately contained in $GTPS$ and define the result of $GTPS$ as follows

Case:

- a) If $GTPS$ immediately contains the <qualified object name> QON , then the result of $GTPS$ is the result of resolving QON with the sequence comprising only $ROPCOR$ as the start object candidate sequence, as specified by General Rule 3) of Subclause 17.8, "<qualified object name>".
- b) Otherwise, the result of $GTPS$ is $ROPCOR$.

- 5) If the <local graph type reference> $LGTR$ is specified, then the result of $LGTR$ is the result of resolving the <qualified graph type name> immediately contained in $LGTR$ with the current working record with appended current working schema as the start object candidate sequence, as specified by General Rule 6).

- 6) If the <qualified graph type name> $QGTN$ is specified, then $QGTN$ is resolved with the start object candidate sequence $SOCSEQ$ as follows:

« WG3:BER-082R1 »

- a) If $QGTN$ immediately contains the <qualified object name> QON , then $SOCSEQ'$ is the sequence comprising the result of resolving QON with $SOCSEQ$ as the start object candidate sequence, as specified by General Rule 3) of Subclause 17.8, "<qualified object name>"; otherwise, $SOCSEQ'$ is $SOCSEQ$.

- b) Let GTN be the <graph type name> immediately contained in $QGTN$ and let RES be the result of resolving GTN with $SOCSEQ'$ as the start object candidate sequence, as specified by General Rule 7).

- c) The result of resolving $QGTN$ with $SOCSEQ$ as the candidate start object sequence is RES .

- 7) If the <graph type name> GTN is specified, then GTN is resolved with the start object candidate sequence $SOCSEQ$ as follows:

- a) Let $NSOCSEQ$ be the number of elements of $SOCSEQ$, let SOC_i , $1 \leq i \leq NSOCSEQ$, be the i -th element of $SOCSEQ$, and let RES_i be determined as follows

Case:

- i) If SOC_i is a GQL-object with named components that contains a graph type GT with name GTN , then RES_i is GT .

- ii) If SOC_i is a GQL-object with named components that contains an alias with name GTN that resolves to a graph type GT , then RES_i is GT .

- iii) Otherwise, RES_i is the null value.

- b) If there is a k , $1 \leq k \leq NSOCSEQ$, such that RES_k is not the null value and such that for every j , $1 \leq j < k$, it holds that RES_j is the null value, then the result of resolving GTN with

the start object candidate sequence $SOCSEQ$ is RES_k ; otherwise, an exception condition is raised:

** Editor's Note (number 202) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

Conformance Rules

None.

17.4 Binding table references

Function

Specify a binding table reference.

Format

```

<binding table reference> ::= 
  <binding table resolution expression>
  | <local binding table reference>

<binding table resolution expression> ::= 
  [ BINDING ] TABLE <catalog binding table reference>

<catalog binding table reference> ::= 
  <catalog binding table parent and name>
  | <predefined table parameter>
  | <external object reference>

<catalog binding table parent and name> ::= 
  <binding table parent specification> <binding table name>
  | <curl path parameter>

<binding table parent specification> ::= 
  [ <parent catalog object reference> ] [ <qualified object name> <period> ]

<local binding table reference> ::= 
  <qualified binding table name>

<qualified binding table name> ::= 
  [ <qualified object name> <period> ] <binding table name>

```

Syntax Rules

**** Editor's Note (number 203) ****

BER-019 has established declared type propagation for working record, working table, and result. The Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-277](#).

- 1) If the <binding table parent specification> *BTPS* does not immediately contain a <parent catalog object reference>, then a <parent catalog object reference> of ./ is implicit.

General Rules

- 1) If the <binding table reference> *BTR* is specified, then the result of *BTR* is the result of the immediately contained <binding table resolution expression> or <local binding table reference>.
- 2) If the <binding table resolution expression> *BTRE* is specified, then the result of *BTRE* is the result of the immediately contained <catalog binding table reference>.
- 3) If the <catalog binding table reference> *CBTR* is specified, then:
 - a) Let *ROCBTR* be determined as follows

Case:

- i) If *CBTR* simply contains the implicit or explicit <binding table parent specification> *BTPS* and the <binding table name> *BTN*, then *ROCBTR* is the result of resolving *BTN* with the sequence comprising only the result of *BTPS* as the start object candidate sequence, as specified by [General Rule 7](#).
 - ii) If *CBTR* immediately contains the <predefined table parameter> *PTP*, then *ROCBTR* is the result of *PTP*.
 - iii) If *CBTR* immediately contains an <external object reference> *EOR*, then the result of *ROCBTR* is the result of *EOR*.
- b) If *ROCBTR* is a binding table, then the result of *CBTR* is *ROCBTR*; otherwise, an exception condition is raised:

** Editor's Note (number 204) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

- 4) If the implicit or explicit <binding table parent specification> *BTPS* is specified, then let *ROPCOR* be the result of the <parent catalog object reference> immediately contained in *BTPS* and define the result of *BTPS* as follows

Case:

« [WG3:BER-082R1](#) »

- a) If *BTPS* immediately contains the <qualified object name> *QON*, then the result of *BTPS* is the result of resolving *QON* with the sequence comprising only *ROPCOR* as the start object candidate sequence, as specified by [General Rule 3](#)) of Subclause 17.8, “<qualified object name>”.
 - b) Otherwise, the result of *BTPS* is *ROPCOR*.
- 5) If the <local binding table reference> *LBTR* is specified, then the result of *LBTR* is the result of resolving the <qualified binding table name> immediately contained in *LBTR* with the current working record with appended current working schema as the start object candidate sequence, as specified by [General Rule 6](#)).
- 6) If the <qualified binding table name> *QBTN* is specified, it is resolved with the start object candidate sequence *SOCSEQ* as follows:

« [WG3:BER-082R1](#) »

- a) If *QBTN* immediately contains the <qualified object name> *QON*, then *SOCSEQ'* is the sequence comprising the result of resolving *QON* with *SOCSEQ* as the start object candidate sequence, as specified by [General Rule 3](#)) of Subclause 17.8, “<qualified object name>”; otherwise, *SOCSEQ'* is *SOCSEQ*.
 - b) Let *BTN* be the <binding table name> immediately contained in *QBTN* and let *RES* be the result of resolving *BTN* with *SOCSEQ'* as the start object candidate sequence, as specified by [General Rule 7](#)).
 - c) The result of resolving *QBTN* with *SOCSEQ* as the candidate start object sequence is *RES*.
- 7) If the <binding table name> *BTN* is specified, *BTN* is resolved with the start object candidate sequence *SOCSEQ* as follows:
- a) Let *NSOCSEQ* be the number of elements of *SOCSEQ*, let *SOC_i*, $1 \leq i \leq NSOCSEQ$, be the *i*-th element of *SOCSEQ*, and let *RES_i* be determined as follows

Case:

17.4 Binding table references

- i) If SOC_i is a GQL-object with named components that contains a binding table BT with name BTN , then RES_i is BT .
 - ii) If SOC_i is a GQL-object with named components that contains an alias with name BTN that resolves to a binding table BT , then RES_i is BT .
 - iii) If SOC_i is a GQL-object with named components that contains a query Q with name BTN that returns a binding table, then RES_i is the binding table obtained by calling Q with no arguments.
 - iv) If SOC_i is a GQL-object with named components that contains an alias with name BTN that resolves to a query Q that returns a binding table, then RES_i is the binding table obtained by calling Q with no arguments.
 - v) Otherwise, RES_i is the null value.
- b) If there is a k , $1 \leq k \leq NSOCSEQ$, such that RES_k is not the null value and such that for every j , $1 \leq j < k$, it holds that RES_j is the null value, then the result of resolving BTN with the start object candidate sequence $SOCSEQ$ is RES_k ; otherwise, an exception condition is raised:

** Editor's Note (number 205) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

Conformance Rules

None.

17.5 Procedure references

Function

Specify a procedure reference.

Format

```

<procedure reference> ::= 
    <procedure resolution expression>
  | <local procedure reference>

<procedure resolution expression> ::= 
  PROCEDURE <catalog procedure reference>

<catalog procedure reference> ::= 
    <catalog procedure parent and name>
  | <external object reference>

<catalog procedure parent and name> ::= 
    <procedure parent specification> <procedure name>
  | <curl path parameter>

<procedure parent specification> ::= 
  [ <parent catalog object reference> ] [ <qualified object name> <period> ]

<local procedure reference> ::= 
  <qualified procedure name>

<qualified procedure name> ::= 
  [ <qualified object name> <period> ] <procedure name>

```

Syntax Rules

- 1) If the <procedure parent specification> *PPS* does not immediately contain a <parent catalog object reference>, then a <parent catalog object reference> of ./ is implicit.

General Rules

- 1) If the <procedure reference> *PR* is specified, then the result of *PR* is the result of the immediately contained <procedure resolution expression> or <local procedure reference>.
- 2) If the <procedure resolution expression> *PRE* is specified, then the result of *PRE* is the result of the immediately contained <catalog procedure reference>.
- 3) If the <catalog procedure reference> *CPR* is specified, then:
 - a) Let *ROCPR* be determined as follows:
 - i) If *CPR* simply contains the implicit or explicit <procedure parent specification> *PPS* and the <procedure name> *PN*, then *ROCPR* is the result of resolving *PN* with the sequence comprising only the result of *PPS* as the start object candidate sequence, as specified by [General Rule 7](#).
 - ii) If *CPR* immediately contains an <external object reference> *EOR*, then the result of *ROCPR* is the result of *EOR*.

- b) If *ROCPR* is a procedure, then the result of *CPR* is *ROCPR*; otherwise, an exception condition is raised:

** Editor's Note (number 206) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

- 4) If the implicit or explicit <procedure parent specification> *PPS* is specified, then let *ROPCOR* be the result of the <parent catalog object reference> immediately contained in *PPS* and define the result of *PPS* as follows

Case:

« WG3:BER-082R1 »

- a) If *PPS* immediately contains the <qualified object name> *QON*, then the result of *PPS* is the result of resolving *QON* with the sequence comprising only *ROPCOR* as the start object candidate sequence, as specified by [General Rule 3](#) of Subclause 17.8, “<qualified object name>”.
 - b) Otherwise, the result of *PPS* is *ROPCOR*.
- 5) If the <local procedure reference> *LPR* is specified, then the result of *LPR* is the result of resolving the <qualified procedure name> immediately contained in *LPR* with the current working record with appended current working schema as the start object candidate sequence, as specified by [General Rule 6](#)).
- 6) If the <qualified procedure name> *QPN* is specified, then *QPN* is resolved with the start object candidate sequence *SOCSEQ* as follows:

« WG3:BER-082R1 »

- a) If *QPN* immediately contains the <qualified object name> *QON*, then *SOCSEQ'* is the sequence comprising the result of resolving *QON* with *SOCSEQ* as the start object candidate sequence, as specified by [General Rule 3](#) of Subclause 17.8, “<qualified object name>”; otherwise, *SOCSEQ'* is *SOCSEQ*.
 - b) Let *PN* be the <procedure name> immediately contained in *QPN* and let *RES* be the result of resolving *PN* with *SOCSEQ'* as the start object candidate sequence, as specified by [General Rule 7](#).
 - c) The result of resolving *QPN* with *SOCSEQ* as the candidate start object sequence is *RES*.
- 7) If the <procedure name> *PN* is specified, then *PN* is resolved with the start object candidate sequence *SOCSEQ* as follows:

- a) Let *NSOCSEQ* be the number of elements of *SOCSEQ*, let *SOC_i*, $1 \leq i \leq NSOCSEQ$, be the *i*-th element of *SOCSEQ*, and let *RES_i* be determined as follows

Case:

- i) If *SOC_i* is a GQL-object with named components that contains a procedure *P* with name *PN*, then *RES_i* is *P*.
- ii) If *SOC_i* is a GQL-object with named components that contains an alias with name *PN* that resolves to a procedure *P*, then *RES_i* is *P*.
- iii) Otherwise, *RES_i* is the null value.

- b) If there is a k , $1 \leq k \leq NSOCSEQ$, such that RES_k is not the null value and such that for every j , $1 \leq j < k$, it holds that RES_j is the null value, then the result of resolving PN with the start object candidate sequence $SOCSEQ$ is RES_k ; otherwise, an exception condition is raised:

** Editor's Note (number 207) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

Conformance Rules

None.

17.6 Function references

Function

Specify a function reference.

Format

```

<function reference> ::==
  <function resolution expression>
  | <local function reference>

<function resolution expression> ::==
  FUNCTION <catalog function reference>

<catalog function reference> ::==
  <catalog function parent and name>
  | <external object reference>

<catalog function parent and name> ::==
  <function parent specification> <function name>
  | <curl path parameter>

<function parent specification> ::==
  [ <parent catalog object reference> ] [ <qualified object name> <period> ]

<local function reference> ::==
  <qualified function name>

<qualified function name> ::==
  [ <qualified object name> <period> ] <function name>

```

Syntax Rules

- 1) If the implicit or explicit <function parent specification> *FPS* does not immediately contain a <parent catalog object reference>, then a <parent catalog object reference> of *. /* is implicit.

General Rules

- 1) If the <function reference> *FR* is specified, then the result of *FR* is the result of the immediately contained <function resolution expression> or <local function reference>.
- 2) If the <function resolution expression> *FRE* is specified, then the result of *FRE* is the result of the immediately contained <catalog function reference>.
- 3) If the <catalog function reference> *CFR* is specified, then:
 - a) Let *ROCFR* be determined as follows

Case:

 - i) If *CFR* simply contains the implicit or explicit <function parent specification> *FPS* and the <function name> *FN*, then *ROCFR* is the result of resolving *FN* with the sequence comprising only the result of *FPS* as the start object candidate sequence, as specified by [General Rule 7](#).
 - ii) If *CFR* immediately contains an <external object reference> *EOR*, then the result of *ROCFR* is the result of *EOR*.

- b) If *ROCFR* is a function, then the result of *CFR* is *ROCFR*; otherwise, an exception condition is raised:

** Editor's Note (number 208) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

- 4) If the implicit or explicit <function parent specification> *FPS* is specified, then let *ROPCOR* be the result of the <parent catalog object reference> immediately contained in *FPS* and define the result of *FPS* as follows

Case:

« WG3:BER-082R1 »

- a) If *FPS* immediately contains the <qualified object name> *QON*, then the result of *FPS* is the result of resolving *QON* with the sequence comprising only *ROPCOR* as the start object candidate sequence, as specified by [General Rule 3](#) of Subclause 17.8, "<qualified object name>".
 - b) Otherwise, the result of *FPS* is *ROPCOR*.
- 5) If the <local function reference> *LFR* is specified, then the result of *LFR* is the result of resolving the <qualified function name> immediately contained in *LFR* with the current working record with appended current working schema as the start object candidate sequence, as specified by [General Rule 6](#).
- 6) If the <qualified function name> *QFN* is specified, then *QFN* is resolved with the start object candidate sequence *SOCSEQ* as follows:

« WG3:BER-082R1 »

- a) If *QFN* immediately contains the <qualified object name> *QON*, then *SOCSEQ'* is the sequence comprising the result of resolving *QON* with *SOCSEQ* as the start object candidate sequence, as specified by [General Rule 3](#) of Subclause 17.8, "<qualified object name>"; otherwise, *SOCSEQ'* is *SOCSEQ*.
 - b) Let *FN* be the <function name> immediately contained in *QFN* and let *RES* be the result of resolving *FN* with *SOCSEQ'* as the start object candidate sequence, as specified by [General Rule 7](#).
 - c) The result of resolving *QFN* with *SOCSEQ* as the candidate start object sequence is *RES*.
- 7) If the <function name> *FN* is specified, then *FN* is resolved with the start object candidate sequence *SOCSEQ* as follows:

- a) Let *NSOCSEQ* be the number of elements of *SOCSEQ*, let *SOC_i*, $1 \leq i \leq NSOCSEQ$, be the *i*-th element of *SOCSEQ*, and let *RES_i* be determined as follows

Case:

- i) If *SOC_i* is a GQL-object with named components that contains a function *F* with name *FN*, then *RES_i* is *F*.
- ii) If *SOC_i* is a GQL-object with named components that contains an alias with name *FN* that resolves to a function *F*, then *RES_i* is *F*.
- iii) Otherwise, *RES_i* is the null value.

- b) If there is a k , $1 \leq k \leq NSOCSEQ$, such that RES_k is not the null value and such that for every j , $1 \leq j < k$, it holds that RES_j is the null value, then the result of resolving FN with the start object candidate sequence $SOCSEQ$ is RES_k ; otherwise, an exception condition is raised:

** Editor's Note (number 209) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

Conformance Rules

None.

17.7 <catalog object reference>

Function

Define a <catalog object reference>.

Format

```

<catalog object reference> ::==
    <catalog url path>

<parent catalog object reference> ::=
    <catalog object reference> [ <solidus> ]

<catalog url path> ::=
    <absolute url path>
    | <relative url path>
    | <parameterized url path>

<absolute url path> ::=
    <solidus> [ <simple url path> ]

<relative url path> ::=
    <parent object relative url path>
    | <simple relative url path>
    | <period>

<parent object relative url path> ::=
    <predefined parent object parameter> [ <solidus> <simple url path> ]

<simple relative url path> ::=
    <double period> [ { <solidus> <double period> }... ] [ <solidus> <simple url path> ]
    | <simple url path>

<parameterized url path> ::=
    <url path parameter> [ <solidus> <simple url path> ]

<simple url path> ::=
    <url segment> [ { <solidus> <url segment> }... ]

<url segment> ::=
    <identifier>

```

**** Editor's Note (number 210) ****

There are differences between the set of allowed characters in a <url segment>. A more detailed analysis is required to better align URL segments and <identifier>s. See Possible Problem [GQL-200](#).

Syntax Rules

- 1) If the <parent catalog object reference> *PCOR* is specified, then either shall *PCOR* only simply contain an <absolute url path> that is <solidus> or *PCOR* shall immediately contain the <solidus>.
- 2) If the <relative url path> *RUP* is specified and *RUP* is <period>, then *RUP* is effectively replaced by CURRENT_SCHEMA.

General Rules

- 1) If the <catalog object reference> *COR* is specified, then the result of *COR* is the result of the <catalog url path> immediately contained in *COR*.
- 2) If the <catalog url path> *CUP* is specified, then its result is defined as follows

Case:

NOTE 136 — The <parameterized url path> is always effectively replaced by the Syntax Rules of Subclause 17.9, “<url path parameter>”.

 - a) If *CUP* is the <absolute url path> *AUP*, then the result of *CUP* is the result of *AUP*.
 - b) If *CUP* is the <relative url path> *RUP*, then the result of *CUP* is the result of *RUP*.
- 3) If the <absolute url path> *AUP* is specified, then

Case:

 - a) If *AUP* immediately contains the <simple url path> *SUP*, then the result of *AUP* is the result of resolving *SUP* with the GQL-catalog root as the resolution start object as specified by General Rule 8).
 - b) Otherwise, the result of *AUP* is the GQL-catalog root.
- 4) If the <relative url path> *RUP* is specified, then:
 - a) If *RUP* immediately contains the <parent object relative url path> *PORUP*, then the result of *RUP* is *PORUP*.
 - b) If *RUP* immediately contains the <simple relative url path> *SRUP*, then the result of *RUP* is *SRUP*.
- 5) If the <parent object relative url path> *PORUP* is specified, then:
 - a) Let *ROPPOP* be the result of the <predefined parent object parameter> immediately contained in *PORUP*.
 - b) If *PORUP* immediately contains the <simple url path> *SUP*, then the result of *PORUP* is the result of resolving *SUP* with *ROPPOP* as the resolution start object, as specified by General Rule 8); otherwise, the result of *PORUP* is *ROPPOP*.
- 6) If the <simple relative url path> *SRUP* is specified, the result of *SRUP* is the result of resolving *SRUP* with the current working schema as the resolution start object, as specified by General Rule 7).
- 7) The result of resolving the <simple relative url path> *SRUP* with the resolution start object *RSO* is defined as follows:
 - a) Let *NDS* be the number of <double period> immediately contained in *RUPT*.
 - b) If *NDS* is larger than the number of parent GQL-objects of *RSO*, then an exception condition is raised: *syntax error or access rule violation — invalid reference (42002)*.
 - c) If *NDS* is 0 (zero), then let *RSO'* be *RSO*; otherwise, let *RSO'* be the *NDS*-th parent GQL-object of *RSO*.
 - d) Case:
 - i) If *SRUP* immediately contains the <simple url path> *SUP*, then the result of *SRUP* is the result of resolving *SUP* with *RSO'* as the resolution start object, as specified by General Rule 7).

- ii) Otherwise, the result of $SRUP$ is RSO' .
- 8) The result of resolving the <simple url path> SUP with the resolution start object RO_0 is defined as follows:
- Let $USSEQ$ be the sequence of all <url segment>s that are immediately contained in SUP , let $NUSSEQ$ be the number of elements of $USSEQ$, and let US_i , $1 \leq i \leq NUSSEQ$, be the i -th element of $USSEQ$.
 - Let the resolved object RO_i , $1 \leq i \leq NUSSEQ$, be determined as follows
- Case:
- If RO_{i-1} is a GQL-object with named components and RO_{i-1} contains a GQL-object X with name US_i , then RO_i is X .
 - Otherwise, an exception condition is raised:

** Editor's Note (number 211) **

Exception condition to be determined. See Possible Problem [GQL-158](#).
- c) The result of resolving SUP with the resolution start object RO_0 is RO_{NUSSEQ} .

Conformance Rules

None.

17.8 <qualified object name>

Function

Define a <qualified object name>.

Format

```
<qualified object name> ::=  
  [ <qualified name prefix> ] <object name>  
  
<qualified name prefix> ::=  
  { <object name> <period> }...
```

Syntax Rules

None.

General Rules

- 1) Let QON be the <qualified object name>.
- 2) QON is object-resolved with the start object candidate sequence $SOCSEQ$ as follows:
« WG3:BER-082R1 deleted one Note »
 - a) Let $NSOCSEQ$ be the number of elements of $SOCSEQ$ and let SOC_i , $1 \leq i \leq NSOCSEQ$, be the i -th element of $SOCSEQ$.
 - b) Let $ONSEQ$ be the sequence of all <object name>s that are simply contained in QON , let $NONSEQ$ be the number of elements of $ONSEQ$, and let ON_j , $1 \leq j \leq NONSEQ$, be the j -th element of $ONSEQ$.
 - c) Let the resolution start object be a GQL-object GO_0 that is determined as follows
 Case:
 - i) If there is a smallest integer k , $1 \leq k \leq NSOCSEQ$, such that SOC_k is a GQL-object with named components that contains a GQL-object with name ON_1 , then GO_0 is SOC_k .
 - ii) Otherwise, an exception condition is raised:

**** Editor's Note (number 212) ****
 Exception condition to be determined. See Possible Problem [\[GQL-158\]](#).
 - d) Let the GQL-object GO_j , $1 \leq j \leq NONSEQ$, be determined as follows
 Case:
 - i) If $j < NONSEQ$ and GO_{j-1} is a GQL-object with named components but not an alias and GO_{j-1} contains a GQL-object X with name ON_j , then GO_j is X .
 - ii) If $j < NONSEQ$ and GO_{j-1} is an alias that successfully resolves to a GQL-object with named components that in turn contains a GQL-object X with name ON_j , then GO_j is X .

- iii) If $j = \text{NONSEQ}$ and GO_{j-1} is a GQL-object with named components and GO_{j-1} contains a GQL-object X with name ON_j , then ON_j is X .
- iv) Otherwise, an exception condition is raised:

** Editor's Note (number 213) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

e) The result of resolving QON with $SOCSEQ$ as the start object candidate sequence is GO_{NONSEQ} .
[« WG3:BER-082R1 »](#)

3) QON is resolved with the start object candidate sequence $SOSSEQ$ as follows:

[« WG3:BER-082R1 deleted one Note »](#)

a) Let $SIMPLE$ be the result of object-resolving QON with $SOCSEQ$ as the start object candidate sequence, as specified by [General Rule 2](#).

[« WG3:BER-082R1 deleted one SR »](#)

[« WG3:BER-082R1 »](#)

b) The result of object-resolving QON with $SOCSEQ$ as the start object candidate sequence is $SIMPLE$.

Conformance Rules

None.

17.9 <url path parameter>

Function

Define a <url path parameter>.

Format

```
<url path parameter> ::=  
  <parameter>
```

Syntax Rules

- 1) Let *UPP* be the <url path parameter>.
- 2) Let *P* be the <parameter> immediately contained in *UPP*.
- 3) The declared type of *P* shall be a character string type.
- 4) Let *ROP* be the result of *P*.

NOTE 137 — The result of *P* is determined from the GQL-request before regular execution and available during the evaluation of Syntax Rules.

- 5) *ROP* shall conform to the Format and Syntax Rules for either an <absolute url path> or a <relative url path>.
- 6) *UPP* is effectively replaced by *ROP*.

General Rules

None.

Conformance Rules

None.

17.10 <external object reference>

Function

Define an <external object reference>.

Format

```
<external object reference> ::=  
  <external object url>  
  
<external object url> ::=  
  !! See the Syntax Rules.
```

Syntax Rules

- 1) Let *EOR* be the <external object reference>.
- 2) Let *EOU* be the <external object url> immediately contained in *EOR*
- 3) *EOU* shall be either an absolute-URL character string or an absolute-URL-with-fragment character string as specified by [URL](#) or it alternatively shall be a URI with a mandatory scheme as specified by [RFC 3986](#) and [RFC 3978](#).
- 4) *EOU* shall not conform to the Format for a <catalog url path>.

General Rules

- 1) The result of *EOR* is the result of resolving *EOU* as specified by the [General Rule 2](#)).
- 2) The result of resolving *EOU* is implementation-defined ([IA010](#)). Determining the result of resolving *EOU* either shall result in a GQL-object or raise an implementation-defined ([ID061](#)) exception.

Conformance Rules

None.

17.11 <element reference>

Function

Reference a graph element or list of graph elements.

Format

```
<element reference> ::=  
  <element variable>
```

Syntax Rules

« WG3:BER-031 »

- 1) Let EV be the <element variable> simply contained in the <element reference> ER . Let MS be the innermost <match statement> that declares EV and that contains ER . Let GP be the <graph pattern> simply contained in MS .
- 2) Let V be the element variable that is identified by EV . ER references V .
- 3) The *degree of reference* of ER is defined as follows

Case:

- a) If ER is simply contained in the <graph pattern where clause> of GT , then the degree of reference of ER is the degree of exposure of EV by GP .

NOTE 138 — “Degree of exposure” is defined in Subclause 16.7, “<path pattern expression>”.

- b) If ER is contained in a <parenthesized path pattern where clause> $PPPWC$ simply contained in GP , then let $PPPE$ be the <parenthesized path pattern expression> that simply contains $PPPWC$.

NOTE 139 — This rule is applied after the syntactic transform that converts any <element pattern where clause> to a <parenthesized path pattern where clause>.

Case:

- i) If EV is declared by $PPPE$, then the degree of reference of ER is the degree of exposure of EV by $PPPE$.

NOTE 140 — “Degree of exposure” is defined in Subclause 16.7, “<path pattern expression>”.

« WG3:BER-031 »

- ii) Otherwise, let PP be the innermost <graph pattern> or <parenthesized path pattern expression> that contains $PPPWC$ and that declares EV . The degree of reference of ER is the degree of exposure of EV by PP . The degree of reference of ER shall be singleton.

NOTE 141 — “Degree of exposure” is defined in Subclause 16.7, “<path pattern expression>”.

General Rules

None.

NOTE 142 — Every <element reference> is evaluated in the General Rules of Subclause 21.5, “Applying bindings to evaluate an expression”.

Conformance Rules

None.

« WG3:BER-031 »

**** Editor's Note (number 214) ****

Subclause <BNF name="path reference"/> not included.

18 Predicates

18.1 <search condition>

Function

Specify a condition that is *True*, *False*, or *Unknown*, depending on the result of a <boolean value expression>.

Format

```
<search condition> ::=  
    <boolean value expression>
```

Syntax Rules

None.

General Rules

- 1) The result of the <search condition> is the result of the <boolean value expression>.
- 2) A <search condition> is said to be *satisfied* if and only if the result of the <boolean value expression> is *True*.

Conformance Rules

None.

18.2 <predicate>

Function

Specify a condition that can be evaluated to give a Boolean value.

Format

```
<predicate> ::=  
  <comparison predicate>  
  | <exists predicate>  
  | <null predicate>  
  | <normalized predicate>  
  | <directed predicate>  
  | <labeled predicate>  
  | <source/destination predicate>  
  | <all_different predicate>  
  | <same predicate>
```

**** Editor's Note (number 215) ****

SQL contains other possibly relevant predicates such as:

- The <in predicate>, which is effectively a syntactic shorthand for a <search condition> involving a <quantified comparison predicate>. The SQL variant allows table subqueries as well as lists, TigerGraph's variant supports bags, but Cypher does not have the syntax although it does have an equivalent for the quantified comparison predicate and a predicate function `none()` which is roughly equivalent to NOT IN.
- The <quantified comparison predicate>, which can operate on arrays and multisets. TigerGraph has no direct equivalent (but see the IN predicate), Cypher has radically different syntax for the same result, it uses the `all()` and `any()` predicate functions as well as the `single()` predicate function, which has no direct equivalent in SQL. Discussion is needed on how to represent this functionality in GQL.
- The <overlaps predicate>, which is syntactic shorthand for a complex <search condition> involving <comparison predicate>s and is very tricky to get right.
- The <member predicate>, which operates on multisets.
- The <submultiset predicate>, which operates on multisets.
- The <set predicate>, which operates on multisets.
- The <type predicate>, which concerns user-defined types. Does this have relevance in the dynamic type context of GQL?
- The <period predicate>, which concerns periods. But see also the <overlaps predicate>.

Discussion on the inclusion or non-inclusion of the above predicates is required.

See Language Opportunity [GQL-177](#).

**** Editor's Note (number 216) ****

SQL contains other predicates that may not be relevant:

- The <exists predicate>, which operates on tables, whereas GQL has an equivalent predicate which operates on graphs.
- The <unique predicate>, which operates on tables.

- The <match predicate> which operates on tables.
- The <distinct predicate>, which operates on rows.

See Language Opportunity [GQL-177](#).

**** Editor's Note (number 217) ****

TigerGraph contains another predicate that may be relevant:

- ISEMPTY, which operates on bags.

Discussion is required as to whether the functions named above should be incorporated into GQL.

See Language Opportunity [GQL-177](#).

Syntax Rules

None.

General Rules

- 1) The result of a <predicate> is the result of the immediately contained <comparison predicate>, <exists predicate>, <null predicate>, <normalized predicate>, <directed predicate>, <labeled predicate>, <source/destination predicate>, <all_different predicate>, or <same predicate>.

Conformance Rules

None.

18.3 <comparison predicate>

** Editor's Note (number 218) **

Comparison of numeric value types having different base types should be further reviewed. See Possible Problem [GQL-255](#).

Function

Specify a comparison of two values.

Format

```
<comparison predicate> ::=  
  <value expression> <comparison predicate part 2>  
  
<comparison predicate part 2> ::=  
  <comp op> <value expression>  
  
<comp op> ::=  
  <equals operator>  
  | <not equals operator>  
  | <less than operator>  
  | <greater than operator>  
  | <less than or equals operator>  
  | <greater than or equals operator>
```

Syntax Rules

** Editor's Note (number 219) **

The rules for comparing collection types have been omitted until collection types have been added to the WD. See Possible Problem [GQL-174](#).

- 1) Let L and R respectively denote the first and second <value expression>s.
- 2) Case:
 - a) If <comp op> is <equals operator> or <not equals operator>, then L and R are operands of an equality operation. The Syntax Rules and Conformance Rules of Subclause 21.8, "Equality operations", apply.
 - b) Otherwise, L and R are operands of an ordering operation. The Syntax Rules and Conformance Rules of Subclause 21.9, "Ordering operations", apply.
- 3) Let CP be the <comparison predicate>. The following syntactic transformations are applied.

Case:

 - a) If the <comp op> is <not equals operator>, then CP is equivalent to:

$$\text{NOT } (L = R)$$
 - b) If the <comp op> is <greater than operator>, then CP is equivalent to:

$$(R < L)$$
 - c) If the <comp op> is <less than or equals operator>, then CP is equivalent to:

$$(L > R)$$

18.3 <comparison predicate>

$$\begin{array}{l} (L < R \\ \text{OR} \\ L = R) \end{array}$$

- d) If the <comp op> is <greater than or equals operator>, then *CP* is equivalent to:

$$\begin{array}{l} (R < L \\ \text{OR} \\ R = L) \end{array}$$

General Rules

- 1) Let *LV* and *RV* be the results of the <value expression>s *L* and *R*, respectively.
- 2) If both *LV* and *RV* are not the null value and the most specific types of the *LV* and *RV* are not comparable, then an exception condition is raised: *data exception — values not comparable (22G04)*.
- 3) Case:
 - a) If at least one of *LV* and *RV* is the null value, then the result of *CP* is *Unknown*.
 - b) If <comp op> is <equals operator>, then:

Case:

 - i) If and only if *LV* and *RV* are equal, then the result of *CP* is *True*.
 - ii) Otherwise, the result of *CP* is *False*.
 - c) If <comp op> is <less than operator>, then:

Case:

 - i) If and only if *LV* is less than *RV*, then the result of *CP* is *True*.
 - ii) Otherwise, the result of *CP* is *False*.
- 4) Numbers are compared with respect to their numeric values.
- 5) The comparison of two character strings is defined as follows:
 - a) If the length in characters of *LV* is not equal to the length in characters of *RV*, then it is implementation-defined (IA015) if the shorter character string is effectively replaced, for the purposes of comparison, with a copy of itself that has been extended to the length of the longer character string by concatenation on the right of one or more <space> characters.
 - b) The Syntax Rules of Subclause 21.10, “Collation determination”, are applied; let *CS* be the *COLL* returned from the application of those Syntax Rules.

** Editor's Note (number 220) **

Consider explicit support for additional collations other than UCS_BASIC and UNICODE (e.g., case insensitive variants, use of alternative collation tables). See Language Opportunity [GQL-012](#).

- c) The result of the comparison of *LV* and *RV* is given by the collation *CS*.
- 6) The comparison of two byte strings is defined as follows:
 - a) Let *LENGTH_LV* be the length in bytes of *LV* and let *LENGTH_RV* be the length in bytes of *RV*. Let $LV_i, 1 \leq i \leq LENGTH_LV$ be the *i*-th byte of *LV*, and let $RV_i, 1 \leq i \leq LENGTH_RV$ be the *i*-th byte of *RV*.

- b) If $LENGTH_LV = LENGTH_RV$ and $LV_i = RV_i$, $1 \leq i \leq LENGTH_LV$, then LV is equal to RV .
 - c) If $LENGTH_LV < LENGTH_RV$, $LV_i = RV_i$ for all $i \leq LENGTH_LV$, and the right-most $LENGTH_RV - LENGTH_LV$ bytes of RV are all X'00's, then it is implementation-defined (IA016) whether LV is equal to RV or whether LV is less than RV .
 - d) If $LENGTH_LV < LENGTH_RV$, $LV_i = RV_i$ for all $i \leq LENGTH_LV$, and at least one of the right-most $LENGTH_RV - LENGTH_LV$ bytes of RV is not X'00's, then LV is less than RV .
 - e) If $LV_j < RV_j$, for some j , $0 < j \leq \min(LENGTH_LV, LENGTH_RV)$, and $LV_i = RV_i$ for all $i < j$, then LV is less than RV .
 - f) Otherwise, LV is greater than RV .
- 7) The comparison of two datetimes is determined according to the duration resulting from their subtraction.
- 8) The comparison of two durations is determined by the comparison of their corresponding values after conversion to a duration expressed in seconds. In the conversion a year is taken to be 365 days and a month 30 days.
- 9) In comparisons of Boolean values, *True* is greater than *False*
«WG3:BER-040R3»
- 10) The comparison of reference values that have different referents but are of the same base type is implementation-defined (IA018).

Conformance Rules

None.

NOTE 143 — If <comp op> is <equals operator> or <not equals operator>, then the Conformance Rules of Subclause 21.8, “Equality operations”, apply; otherwise, the Conformance Rules of Subclause 21.9, “Ordering operations”, apply.

18.4 <exists predicate>

Function

Specify an existential subquery.

Format

```
<exists predicate> ::=  
  EXISTS {  
    <left paren> <graph pattern> <right paren>  
  | <nested query specification>  
 }
```

Syntax Rules

- 1) Let *EP* be the <exists predicate>.
- 2) If *EP* immediately contains the <graph pattern> *GP*, *EP* is effectively replaced by:

```
EXISTS { MATCH GP RETURN * }
```

General Rules

- 1) Let *NQS* be the <nested query specification> immediately contained in *EP*.
- 2) In a new child execution context:
 - a) The General Rules of *NQS* are applied.
 - b) Case:
 - i) If the current execution result is a non-empty graph, the result of *EP* is *True*.
 - ii) If the current execution result is an empty graph, the result of *EP* is *False*.
 - iii) If the current execution result is a non-empty table, the result of *EP* is *True*.
 - iv) If the current execution result is an empty table, the result of *EP* is *False*.
 - v) If the current execution result is a non-empty collection value, the result of *EP* is *True*.
 - vi) If the current execution result is an empty collection value, the result of *EP* is *False*.
 - vii) If the current execution result is a non-empty map or struct, the result of *EP* is *True*.
 - viii) If the current execution result is an empty map or struct, the result of *EP* is *False*.
 - ix) If the current execution result is *True*, the result of *EP* is *True*.
 - x) If the current execution result is *False*, the result of *EP* is *False*.
 - xi) If the current execution result is the null value, the result of *EP* is *False*.
 - xii) Otherwise, an exception condition is raised:

** Editor's Note (number 221) **

Exception condition to be determined. See Possible Problem [GQL-158](#).

Conformance Rules

None.

18.5 <null predicate>

Function

Specify a test for a null value.

Format

```
<null predicate> ::=  
  <value expression primary> <null predicate part 2>  
  
<null predicate part 2> ::=  
  IS [ NOT ] NULL
```

Syntax Rules

None.

General Rules

- 1) Let R be the <value expression primary> and let V be the result of R .
- 2) Case:
 - a) If V is the null value, then " R IS NULL" is True and the result of " R IS NOT NULL" is False.
 - b) Otherwise, " R IS NULL" is False and the result of " R IS NOT NULL" is True.

Conformance Rules

None.

18.6 <normalized predicate>

Function

Determine whether a character string value is normalized.

Format

```
<normalized predicate> ::=  
  <string value expression> <normalized predicate part 2>  
  
<normalized predicate part 2> ::=  
  IS [ NOT ] [ <normal form> ] NORMALIZED
```

Syntax Rules

- 1) Let <string value expression> be *SVE*.
- 2) Case:
 - a) If <normal form> is specified, then let *NF* be <normal form>.
 - b) Otherwise, let *NF* be NFC.
- 3) The expression

SVE IS NOT NF NORMALIZED

is equivalent to:

NOT (*SVE IS NF NORMALIZED*)

General Rules

- 1) The result of *SVE IS NF NORMALIZED* is
Case:
 - a) If the result of *SVE* is the null value, then *Unknown*.
 - b) If the result of *SVE* is in the normalization form specified by *NF*, as defined by [Unicode Standard Annex #15](#), then *True*.
 - c) Otherwise, *False*.

Conformance Rules

None.

18.7 <directed predicate>

Function

Determine whether an edge variable is bound to a directed edge.

Format

```
<directed predicate> ::=  
  <element reference> <directed predicate part 2>  
  
<directed predicate part 2> ::=  
  IS [ NOT ] DIRECTED
```

Syntax Rules

- 1) Let *ER* be the <element reference>. *ER* shall have singleton degree of reference, and shall reference an edge variable.
- 2) If NOT is specified, then the <directed predicate> is equivalent to:

NOT (*ER* IS DIRECTED)

General Rules

- 1) Let *LOE* be the list of elements that are bound to *ER*.
- 2) The result of the <directed predicate>

ER IS DIRECTED

is defined as follows

Case:

- a) If *LOE* is empty, then Unknown.
- b) If the sole graph element in *LOE* is a directed edge, then True.
- c) Otherwise, False.

Conformance Rules

- 1) Without Feature G110, “IS DIRECTED predicate”, conforming GQL language shall not contain a <directed predicate>.

18.8 <labeled predicate>

Function

Determine whether a graph element satisfies a <label expression>.

Format

```
<labeled predicate> ::=  
  <element reference> <labeled predicate part 2>  
  
<labeled predicate part 2> ::=  
  <is labeled or colon> <label expression>  
  
<is labeled or colon> ::=  
  IS [ NOT ] LABELED  
  | <colon>
```

Syntax Rules

- 1) Let *ER* be the <element reference>. *ER* shall have singleton degree of reference.
- 2) Let *LE* be the <label expression>.
- 3) If NOT is specified, then the <labeled predicate> is equivalent to:

$$\text{NOT} (\text{ } \text{ER} \text{ } \text{IS} \text{ } \text{LABLED} \text{ } \text{LE} \text{ } \text{)}$$
- 4) If <is labeled or colon> *ILOC* is <colon>, then *ILOC* is replaced by IS LABELED.

General Rules

- 1) Let *LOE* be the list of elements that are bound to *ER*.
- 2) The result of the <labeled predicate> *LP*

ER IS LABELED *LE*

is defined as follows

Case:

- a) If *LOE* is empty, then the value of *LP* is *Unknown*.
- b) Otherwise, let *E* be the sole graph element in *LOE*. Let *LS* be the label set of *E*. The Syntax Rules of Subclause 21.1, “Satisfaction of a <label expression> by a label set”, are applied with *LE* as *LABEL EXPRESSION* and *LS* as *LABEL SET*; let *TV* be the *TRUTH VALUE* returned from the application of those Syntax Rules. The value of *LP* is *TV*.

Conformance Rules

- 1) Without Feature G111, “IS LABELED predicate”, conforming GQL language shall not contain a <labeled predicate>.

18.9 <source/destination predicate>

Function

Determine whether a node is the source or destination of an edge.

Format

```

<source/destination predicate> ::= 
  <node reference> <source predicate part 2>
  | <node reference> <destination predicate part 2>

<node reference> ::= 
  <element reference>

<source predicate part 2> ::= 
  IS [ NOT ] SOURCE [ OF ] <edge reference>

<destination predicate part 2> ::= 
  IS [ NOT ] DESTINATION [ OF ] <edge reference>

<edge reference> ::= 
  <element reference>

```

Syntax Rules

- 1) Let *NR* be the <node reference>. *NR* shall have singleton degree of reference, and shall reference a node variable.
- 2) Let *ER* be the <edge reference>. *ER* shall have singleton degree of reference, and shall reference an edge variable.
- 3) Let *SOD* be the <key word> SOURCE or DESTINATION simply contained in the <source/destination predicate>.
- 4) If NOT is specified, then the <source/destination predicate> is equivalent to:

`NOT (VR IS SOD OF ER)`

General Rules

- 1) Let *LOV* be the list of elements that are bound to *VR* and let *LOE* be the list of elements that are bound to *ER*.
- 2) The result of the <source/destination predicate> *SDP*

`VR IS SOD OF ER`

is defined as follows

Case:

- a) If *LOV* is empty or if *LOE* is empty, then the value of *SDP* is Unknown.
- b) Otherwise, let *V* be the sole graph element in *LOV*, and let *E* be the sole graph element in *LOE*.

Case:

- i) If E is an undirected edge, then the value of SDP is *False*.
- ii) If SOD is SOURCE and V is the source node of E , then the value of SDP is *True*.
- iii) If SOD is DESTINATION and V is the destination node of E , then the value of SDP is *True*.
- iv) Otherwise, the value of SDP is *False*.

Conformance Rules

- 1) Without Feature G112, “IS SOURCE and IS DESTINATION predicate”, conforming GQL language shall not contain a <source/destination predicate>.

18.10 <all_different predicate>

Function

Determine whether all graph elements bound to a list of element references are pairwise different from one another.

Format

```
<all_different predicate> ::=  
  ALL_DIFFERENT <left paren> <element reference> <comma> <element reference>  
    [ { <comma> <element reference> } ... ]  
  <right paren>
```

Syntax Rules

- 1) Each <element reference> simply contained in <all_different predicate> *ADP* shall have unconditional singleton degree of reference.

General Rules

- 1) Let N be the number of <element reference>s simply contained in *ADP*, and let ER_1, \dots, ER_N be an enumeration of those <element reference>s.
- 2) For every i , $1 \leq i \leq N$, let LOE_i be the list of graph elements of ER_i , and let GE_i be the sole graph element in LOE_i .
- 3) The value of *ADP* is
Case:
 - a) If there exist j, k such that $j < k$ and GE_j is the same graph element as GE_k , then *False*.
 - b) Otherwise, *True*.

Conformance Rules

- 1) Without Feature G113, “ALL_DIFFERENT predicate”, conforming GQL language shall not contain an <all_different predicate>.

18.11 <same predicate>

Function

Determine whether all element references in a list of element references bind to the same graph element.

Format

```
<same predicate> ::=  
  SAME <left paren> <element reference> <comma> <element reference>  
    [ { <comma> <element reference> }... ]  
  <right paren>
```

Syntax Rules

- 1) Each <element reference> simply contained in <same predicate> *SP* shall have unconditional singleton degree of reference.

General Rules

- 1) Let *N* be the number of <element reference>s simply contained in *SP*, and let *ER*₁, ..., *ER*_{*N*} be an enumeration of those <element reference>s.
- 2) For every *i*, 1 (one) ≤ *i* ≤ *N*, let *LOE*_{*i*} be the list of graph elements of *ER*_{*i*}, and let *GE*_{*i*} be the sole graph element in *LOE*_{*i*}.
- 3) The value of *SP* is
Case:
 - a) If every *GE*_{*i*} is the same graph element, then *True*.
 - b) Otherwise, *False*.

Conformance Rules

- 1) Without Feature G114, “SAME predicate”, conforming GQL language shall not contain a <same predicate>.

19 Value expressions

19.1 <value specification>

Function

Specify one or more values or parameters.

Format

```

<value specification> ::=

  <literal>
  | <parameter value specification>

<unsigned value specification> ::=

  <unsigned literal>
  | <parameter value specification>

<unsigned integer specification> ::=

  <unsigned integer>
  | <parameter>

<parameter value specification> ::=

  <parameter>
  | <predefined parameter>

<predefined parameter> ::=

  <predefined parent object parameter>
  | <predefined table parameter>
  | CURRENT_USER

<predefined parent object parameter> ::=

  <predefined schema parameter>
  | <predefined graph parameter>

<predefined schema parameter> ::=

  HOME_SCHEMA
  | CURRENT_SCHEMA

<predefined graph parameter> ::=

  EMPTY_PROPERTY_GRAPH
  | EMPTY_GRAPH
  | HOME_PROPERTY_GRAPH
  | HOME_GRAPH
  | CURRENT_PROPERTY_GRAPH
  | CURRENT_GRAPH

<predefined table parameter> ::=

  EMPTY_BINDING_TABLE
  | EMPTY_TABLE
  | UNIT_BINDING_TABLE
  | UNIT_TABLE

```

** Editor's Note (number 222) **

The graph-type specific expressions needed by GQL need to be fully defined. See Possible Problem [GQL-008](#).

Syntax Rules

- 1) The declared type of CURRENT_USER is character string.
- 2) The declared type of CURRENT_SCHEMA and HOME_SCHEMA is the schema type.
- 3) The declared type of EMPTY_GRAPH, EMPTY_PROPERTY_GRAPH, CURRENT_GRAPH, CURRENT_PROPERTY_GRAPH, HOME_GRAPH, and HOME_PROPERTY_GRAPH is a graph type of the specified graph. This graph type shall be a graph type supported by the GQL-implementation and is either the graph type specified explicitly by the user when creating the graph or in the absence of having been specified by the user inferred in an implementation-defined [\(IW008\)](#) way.
- 4) The declared type of EMPTY_TABLE, EMPTY_BINDING_TABLE, UNIT_TABLE, and UNIT_BINDING_TABLE is the most specific binding table type of the specified binding table. This binding table type shall be a binding table type supported by the GQL-implementation and is inferred in an implementation-defined [\(IW009\)](#) way.
- 5) The declared type of <unsigned integer specification> shall be an implementation-defined [\(ID062\)](#) integer type.

General Rules

- 1) A <value specification> or <unsigned value specification> specifies a value that is not selected from a graph.
- 2) A <parameter name> identifies a session parameter or a request parameter.
- 3) The value specified by a <literal> is the value represented by that <literal>.
- 4) The object specified by EMPTY_BINDING_TABLE and EMPTY_TABLE is a table with no rows and no columns.
- 5) The object specified by EMPTY_GRAPH and EMPTY_PROPERTY_GRAPH is a graph that has no nodes, edges, labels, and properties.
- 6) The value specified by CURRENT_USER is
Case:
 - a) If there is a current authorization identifier, then the value of that current authorization identifier.
 - b) Otherwise, the null value.
- 7) The value specified by HOME_SCHEMA is the current home schema.
- 8) The value specified by HOME_PROPERTY_GRAPH or by HOME_GRAPH is the current home graph.
- 9) The value specified by CURRENT_SCHEMA is the current working schema.
- 10) The value specified by CURRENT_PROPERTY_GRAPH or by CURRENT_GRAPH is the current working graph.
- 11) The object specified by UNIT_BINDING_TABLE and UNIT_TABLE is a table with exactly one record and no columns.

- 12) If a <value specification> evaluates to the null value, then an exception condition is raised: *data exception — null value not allowed (22004)*.
- 13) If the most specific type of <unsigned integer specification> is not an exact numeric with scale 0 (zero) then an exception condition is raised: *data exception — invalid value type (22G03)*.
- 14) Let V be the result of <unsigned integer specification>.

Case:

- a) If V is the null value, then an exception condition is raised: *data exception — null value not allowed (22004)*.
- b) If V is negative, then an exception condition is raised: *data exception — negative limit value (22G02)*.

Conformance Rules

None.

19.2 <value expression>

Function

Specify a constant or a value.

Format

```
« WG3:BER-040R3 »  
« WG3:BER-094R1 »  
  
<value expression> ::=  
    <common value expression>  
    | <boolean value expression>  
« WG3:BER-094R1 »  
  
<common value expression> ::=  
    <numeric value expression>  
    | <string value expression>  
    | <datetime value expression>  
    | <duration value expression>  
    | <list value expression>  
    | <map value expression>  
    | <record value expression>  
    | <reference value expression>  
« Consequence of WG3:BER-040R3 »  
  
<reference value expression> ::=  
    <node reference value expression>  
    | <edge reference value expression>  
  
<node reference value expression> ::=  
    <value expression primary>  
  
<edge reference value expression> ::=  
    <value expression primary>  
« WG3:BER-094R1 deleted two productions »  
« WG3:BER-038R1 deleted one production »  
  
<map value expression> ::=  
    <value expression primary>
```

** Editor's Note (number 223) **

Further detail needs to be added regarding <map value expression>. See Possible Problem
[GQL-083](#).

```
<record value expression> ::=  
    <value expression primary>  
« WG3:BER-040R3 deleted one Editor's Note (not in paper) »  
« WG3:BER-094R1 deleted one Editor's Note »
```

Syntax Rules

« WG3:BER-094R1 »

- 1) The declared type of a <node reference value expression> shall be a node reference type.

- 2) The declared type of an <edge reference value expression> shall be an edge reference type.
- 3) The declared type of a <map value expression> shall be a map value type.
- 4) The declared type of a <record value expression> shall be a record type.

General Rules

« WG3:BER-094R1 deleted thirteen GRs »

« WG3:BER-038R1 deleted two GRs »

None.

Conformance Rules

None.

19.3 <boolean value expression>

Function

Specify a Boolean value.

Format

```

<boolean value expression> ::=

  <boolean term>
  | <boolean value expression> OR <boolean term>
  | <boolean value expression> XOR <boolean term>

<boolean term> ::=

  <boolean factor>
  | <boolean term> AND <boolean factor>

<boolean factor> ::=

  [ NOT ] <boolean test>

<boolean test> ::=

  <boolean primary> [ {
    IS [ NOT ]
  | <equals operator>
  | <not equals operator>
  } <truth value> ]

« WG3:BER-094R1 »

<truth value> ::=

  TRUE
  | FALSE
  | UNKNOWN

<boolean primary> ::=

  <predicate>
  | <boolean predicand>

<boolean predicand> ::=

  <parenthesized boolean value expression>
  | <non-parenthesized value expression primary>

<parenthesized boolean value expression> ::=
  <left paren> <boolean value expression> <right paren>

```

Syntax Rules

**** Editor's Note (number 224) ****

Rules regarding known-not null are ignored for the moment. See Possible Problem [GQL-018](#).

**** Editor's Note (number 225) ****

Rules regarding known-not null and determinism are ignored for the moment. See Language Opportunity [GQL-011](#).

« WG3:BER-094R1 »

- 1) The declared type of a <boolean value expression> is a Boolean type.

19.3 <boolean value expression>

- 2) The declared type of a <parenthesized boolean value expression> is the declared type of the simply contained <boolean value expression>.
- 3) The declared type of a <boolean predicand> that is a <non-parenthesized value expression primary> shall be a Boolean type.
- 4) $X \text{ XOR } Y$ is equivalent to the following <boolean value expression>:

(X OR Y) AND NOT (X AND Y)

« WG3:BER-094R1 deleted one SR »

- 5) If NOT is specified in a <boolean test>, then let BP be the contained <boolean primary> and let TV be the contained <truth value>. The <boolean test> is equivalent to:

(NOT (BP IS TV))

General Rules

« WG3:BER-094R1 deleted one GR »

- 1) The result is derived by the application of the specified Boolean operators ("AND", "OR", "NOT", and "IS") to the results derived from each <boolean primary>. If Boolean operators are not specified, then the result of the <boolean value expression> is the result of the specified <boolean primary>.
- 2) NOT (True) is False, NOT (False) is True, and NOT (Unknown) is Unknown.
- 3) Table 3, "Truth table for the AND Boolean operator", Table 4, "Truth table for the OR Boolean operator", and Table 5, "Truth table for the IS Boolean operator", specify the semantics of AND, OR, and IS, respectively.

Table 3 — Truth table for the AND Boolean operator

AND	<u>True</u>	<u>False</u>	<u>Unknown</u>
<u>True</u>	<u>True</u>	<u>False</u>	<u>Unknown</u>
<u>False</u>	<u>False</u>	<u>False</u>	<u>False</u>
<u>Unknown</u>	<u>Unknown</u>	<u>False</u>	<u>Unknown</u>

Table 4 — Truth table for the OR Boolean operator

OR	<u>True</u>	<u>False</u>	<u>Unknown</u>
<u>True</u>	<u>True</u>	<u>True</u>	<u>True</u>
<u>False</u>	<u>True</u>	<u>False</u>	<u>Unknown</u>
<u>Unknown</u>	<u>True</u>	<u>Unknown</u>	<u>Unknown</u>

Table 5 — Truth table for the IS Boolean operator

IS	TRUE	FALSE	UNKNOWN
<i>True</i>	<i>True</i>	<i>False</i>	<i>False</i>
<i>False</i>	<i>False</i>	<i>True</i>	<i>False</i>
<i>Unknown</i>	<i>False</i>	<i>False</i>	<i>True</i>

Conformance Rules

None.

19.4 <numeric value expression>

Function

Specify a numeric value.

Format

```

<numeric value expression> ::=

  <term>
  | <numeric value expression> <plus sign> <term>
  | <numeric value expression> <minus sign> <term>

<term> ::=

  <factor>
  | <term> <asterisk> <factor>
  | <term> <solidus> <factor>

<factor> ::=

  [ <sign> ] <numeric primary>

<numeric primary> ::=

  <value expression primary>
  | <numeric value function>

```

Syntax Rules

**** Editor's Note (number 226) ****

[BER-019] and [BER-094R1] established declared type propagation for the result. This Subclause does not do that yet and requires corresponding Syntax Rules. See [Possible Problem GQL-289](#).

- 1) If a <numeric value expression> immediately contains a <minus sign> *NMS* and immediately contains a <term> that is a <factor> that immediately contains a <sign> that is a <minus sign> *FMS*, then there shall be a <separator> between *NMS* and *FMS*.

General Rules

- 1) The declared type of a <factor> is that of the immediately contained <numeric primary>.
- 2) If the most specific type of a <numeric primary> is not numeric, then an exception is raised: *data exception — invalid value type (22G03)*.
- 3) Case:
 - a) If the most specific type of either operand of a dyadic arithmetic operator is approximate numeric, then the declared type of the result is an implementation-defined ([ID063](#)) approximate numeric type.
 - b) Otherwise, the declared type of both operands of a dyadic arithmetic operator is exact numeric and the declared type of the result is an implementation-defined ([ID064](#)) exact numeric type, with precision and scale defined as follows:
 - i) Let *S1* and *S2* be the scale of the first and second operands respectively.
 - ii) The precision of the result of addition and subtraction is implementation-defined ([ID065](#)), and the scale is the maximum of *S1* and *S2*.

- iii) The precision of the result of multiplication is implementation-defined (ID066), and the scale is $S1 + S2$.
 - iv) The precision and scale of the result of division are implementation-defined (ID067).
- 4) If the result of any <numeric primary> simply contained in a <numeric value expression> is the null value, then the result of the <numeric value expression> is the null value.
 - 5) If the <numeric value expression> contains only a <numeric primary>, then the result of the <numeric value expression> is the result of the specified <numeric primary>.
 - 6) The monadic arithmetic operators <plus sign> and <minus sign> (+ and -, respectively) specify monadic plus and monadic minus, respectively. Monadic plus does not change its operand. Monadic minus reverses the sign of its operand.
 - 7) The dyadic arithmetic operators <plus sign>, <minus sign>, <asterisk>, and <solidus> (+, -, *, and /, respectively) specify addition, subtraction, multiplication, and division, respectively. If the value of a divisor is zero, then an exception condition is raised: *data exception — division by zero* (22012).
 - 8) If the most specific type of the result of an arithmetic operation is exact numeric, then
 - Case:
 - a) If the operator is not division and the mathematical result of the operation is not exactly representable with the precision and scale of the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range* (22003).
 - b) If the operator is division and the approximate mathematical result of the operation represented with the precision and scale of the declared type of the result loses one or more leading significant digits after rounding or truncating if necessary, then an exception condition is raised: *data exception — numeric value out of range* (22003). The choice of whether to round or truncate is implementation-defined (IA011).
 - 9) If the most specific type of the result of an arithmetic operation is approximate numeric and the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range* (22003).

Conformance Rules

None.

19.5 <value expression primary>

Function

Specify a value that is syntactically self-delimited.

Format

```

<value expression primary> ::= 
  <parenthesized value expression>
  | <non-parenthesized value expression primary>

<parenthesized value expression> ::= 
  <left paren> <value expression> <right paren>

<non-parenthesized value expression primary> ::= 
  <property reference>
  | <binding variable>
  | <parameter value specification>
  | <unsigned value specification>
  | <aggregate function>
  | <collection value constructor>
  | <value query expression>
  | <case expression>
  | <cast specification>
  | <element_id function>

« WG3:BER-094R1 »

<collection value constructor> ::= 
  <list value constructor>
  | <map value constructor>
  | <record value constructor>

```

** Editor's Note (number 227) **

It needs to be decided if evaluating aggregate functions over the current working table should be supported by occurrence of a <value expression>. See Language Opportunity [GQL-017](#).

Syntax Rules

None.

General Rules

« WG3:BER-094R1 deleted two GRs »

None.

Conformance Rules

None.

19.6 <numeric value function>

Function

Specify a function yielding a value of type numeric.

Format

```

<numeric value function> ::=

    <length expression>
  | <absolute value expression>
  | <modulus expression>
  | <trigonometric function>
  | <general logarithm function>
  | <common logarithm>
  | <natural logarithm>
  | <exponential function>
  | <power function>
  | <square root>
  | <floor function>
  | <ceiling function>
  | <inDegree function>
  | <outDegree function>

<length expression> ::=
    <char length expression>
  | <byte length expression>
  | <path length expression>

<char length expression> ::=
  CHARACTER_LENGTH <left paren> <character string value expression> <right paren>

<byte length expression> ::=
{
  BYTE_LENGTH
  | OCTET_LENGTH
} <left paren> <string value expression> <right paren>

<path length expression> ::=
  LENGTH <left paren> <binding variable> <right paren>

<absolute value expression> ::=
  ABS <left paren> <numeric value expression> <right paren>

<modulus expression> ::=
  MOD <left paren> <numeric value expression dividend> <comma>
    <numeric value expression divisor> <right paren>

<numeric value expression dividend> ::=
  <numeric value expression>

<numeric value expression divisor> ::=
  <numeric value expression>

<trigonometric function> ::=
  <trigonometric function name> <left paren> <numeric value expression> <right paren>

<trigonometric function name> ::=
  SIN | COS | TAN | COT | SINH | COSH | TANH | ASIN | ACOS | ATAN | DEGREES | RADIANS

<general logarithm function> ::=

```

19.6 <numeric value function>

```

LOG <left paren> <general logarithm base> <comma>
      <general logarithm argument> <right paren>

<general logarithm base> ::= 
    <numeric value expression>

<general logarithm argument> ::= 
    <numeric value expression>

<common logarithm> ::= 
    LOG10 <left paren> <numeric value expression> <right paren>

<natural logarithm> ::= 
    LN <left paren> <numeric value expression> <right paren>

<exponential function> ::= 
    EXP <left paren> <numeric value expression> <right paren>

<power function> ::= 
    POWER <left paren> <numeric value expression base> <comma>
        <numeric value expression exponent> <right paren>

<numeric value expression base> ::= 
    <numeric value expression>

<numeric value expression exponent> ::= 
    <numeric value expression>

<square root> ::= 
    SQRT <left paren> <numeric value expression> <right paren>

<floor function> ::= 
    FLOOR <left paren> <numeric value expression> <right paren>

<ceiling function> ::= 
    { CEIL | CEILING } <left paren> <numeric value expression> <right paren>

<inDegree function> ::= 
    inDegree <left paren> <binding variable> <right paren>

<outDegree function> ::= 
    outDegree <left paren> <binding variable> <right paren>

```

**** Editor's Note (number 228) ****

SQL contains other possibly relevant numeric value functions such as:

- 1) POSITION for character strings and byte strings.
- 2) EXTRACT for datetimes and durations. Cypher has a range of functions with similar functionality.
- 3) CARDINALITY for collections. See also Cypher function size().

See Language Opportunity [GQL-176](#).

**** Editor's Note (number 229) ****

SQL contains other numeric value functions that are probably not relevant such as:

- 1) WIDTH_BUCKET.
- 2) MATCH_NUMER
- 3) ARRAY_MAX_CARDINALITY.

See Language Opportunity [GQL-176](#).

**** Editor's Note (number 230) ****

Cypher contains other possibly relevant numeric value functions such as:

- 1) size() - returns the number of items in a list, the length of a character string, or the number of subgraphs matching the pattern expression. See also the SQL function CARDINALITY and the current GQL function CHARACTER_LENGTH.
- 2) round() - returns the value of a number rounded to the nearest integer.
- 3) rand() - returns a random floating point number in the range from 0 (inclusive) to 1 (exclusive); i.e., [0,1). The numbers returned follow an approximate uniform distribution.
- 4) e() - returns the base of the natural logarithm, e.
- 5) sign() - returns an indication of the sign of a number: 0 if the number is 0, -1 for any negative number, and 1 for any positive number.
- 6) pi() - returns the mathematical constant pi.
- 7) haversin() - returns half the versed sine of a number.
- 8) atan2() - returns the arctangent2 of a set of coordinates in radians.
- 9) reduce() - returns the value resulting from the application of an expression on each successive element in a list in conjunction with the result of the computation thus far. This function iterates through each element e in the given list, run the expression on e - taking into account the current partial result - and store the new partial result in the accumulator. This function is analogous to the fold or reduce method in functional languages such as Lisp and Scala.

See Language Opportunity [GQL-176](#).

**** Editor's Note (number 231) ****

Cypher contains other numeric value functions that may not be relevant such as:

- 1) id() - returns the id of a relationship or node.

See Language Opportunity [GQL-176](#).

**** Editor's Note (number 232) ****

Discussion is required as to whether any of the functions named above should be incorporated into GQL. See Language Opportunity [GQL-176](#).

Syntax Rules

**** Editor's Note (number 233) ****

[BER-019] and [BER-094R1] established declared type propagation for the result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-290](#).

- 1) If <common logarithm> is specified, then let *NVE* be the simply contained <numeric value expression>. The <common logarithm> is equivalent to:

`LOG (10, NVE)`

- 2) If <square root> is specified, then let NVE be the simply contained <numeric value expression>. The <square root> is equivalent to:

```
POWER (NVE, 0.5)
```

General Rules

- 1) If a <length expression> is specified, then the declared type of the result is an implementation-defined (ID068) exact numeric type with scale 0 (zero).
- 2) If <path length expression> is specified and the immediately contained <binding variable> does not identify a path then an exception is raised: *data exception — invalid value type (22G03)*.
- 3) If <absolute value expression> is specified, then the declared type of the result is the declared type of the immediately contained <numeric value expression>.
- 4) If <trigonometric function> is specified, then the declared type of the result is an implementation-defined (ID069) approximate numeric type.
- 5) The declared type of the result of <general logarithm function> is an implementation-defined (ID070) approximate numeric type.
- 6) The declared type of the result of <natural logarithm> is an implementation-defined (ID071) approximate numeric type.
- 7) The declared type of the result of <exponential function> is an implementation-defined (ID072) approximate numeric type.
- 8) The declared type of the result of <power function> is an implementation-defined (ID073) approximate numeric type.
- 9) If <floor function> or <ceiling function> is specified, then

Case:

- a) If the declared type of the immediately contained <numeric value expression> NVE is exact numeric, then the declared type of the result is exact numeric with implementation-defined (ID074) precision, with the radix of NVE , and with scale 0 (zero).
- b) Otherwise, an approximate numeric with implementation-defined (ID075) precision.

- 10) If <absolute value expression> is specified, then let N be the result of the immediately contained <numeric value expression>.

Case:

- a) If N is the null value, then the result is the null value.
- b) If $N \geq 0$, then the result is N .
- c) Otherwise, the result is $-1 * N$. If $-1 * N$ is not representable by the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range (22003)*.

- 11) If <modulus expression> is specified, then let N be the value of the immediately contained <numeric value expression dividend> and let M be the value of the immediately contained <numeric value expression divisor>.

Case:

- a) If at least one of N and M is the null value, then the result is the null value.

- b) If M is zero, then an exception condition is raised: *data exception — division by zero (22012)*.
 - c) Otherwise, the result is the unique exact numeric value R with scale 0 (zero) such that all of the following are true:
 - i) R has the same sign as N .
 - ii) The absolute value of R is less than the absolute value of M .
 - iii) $N = M * K + R$ for some exact numeric value K with scale 0 (zero).
- 12) If <trigonometric function> is specified, then let V be the result of the immediately contained <numeric value expression> that represents an angle expressed in radians.
- Case:
- a) If V is the null value, then the result is the null value.
 - b) Otherwise, let OP be the <trigonometric function name>.
- Case:
- i) If OP is ACOS, then
- Case:
- 1) If V is less than -1 (negative one) or V is greater than 1 (one), then an exception condition is raised: *data exception — numeric value out of range (22003)*.
 - 2) Otherwise, the result of the <trigonometric function> is the inverse cosine of V .
- ii) If OP is ASIN, then
- Case:
- 1) If V is less than -1 (negative one) or V is greater than 1 (one), then an exception condition is raised: *data exception — numeric value out of range (22003)*.
 - 2) Otherwise, the result of the <trigonometric function> is the inverse sine of V .
- iii) If OP is ATAN, then the result is the inverse tangent of V .
 - iv) If OP is COS, then the result is the cosine of V .
 - v) If OP is COSH, then the result is the hyperbolic cosine of V .
 - vi) If OP is SIN, then the result is the sine of V .
 - vii) If OP is SINH, then the result is the hyperbolic sine of V .
 - viii) If OP is TAN, then the result is the tangent of V .
 - ix) If OP is TANH, then the result is the hyperbolic tangent of V .
 - x) If OP is COT, then the result is the cotangent of V .
 - xi) If OP is DEGREES, then the result is the value of V , taken to be in radians, expressed as degrees.
 - xii) If OP is RADIANS, then the result is the value of V , taken to be in degrees, expressed as radians.
- 13) If <general logarithm function> is specified, then let VB be the value of the <general logarithm base> and let VA be the value of the <general logarithm argument>.

Case:

- a) If at least one of VA and VB is the null value, then the result is the null value.
- b) If VA is negative or 0 (zero), then an exception condition is raised: *data exception — numeric value out of range (22003)*.
- c) If VB is negative, 0 (zero), or 1 (one), then an exception condition is raised: *data exception — numeric value out of range (22003)*.
- d) Otherwise, the result is the logarithm with base VB of VA .

14) If <natural logarithm> is specified, then let V be the value of the immediately contained <numeric value expression>.

Case:

- a) If V is the null value, then the result is the null value.
- b) If V is 0 (zero) or negative, then an exception condition is raised: *data exception — invalid argument for natural logarithm (2201E)*.
- c) Otherwise, the result is the natural logarithm of V .

15) If <exponential function> is specified, then let V be the result of the immediately contained <numeric value expression>.

Case:

- a) If V is the null value, then the result is the null value.
- b) Otherwise, the result is e (the base of natural logarithms) raised to the power V . If the result is not representable in the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range (22003)*.

16) If <power function> is specified, then let $NVEB$ be the <numeric value expression base>, then let VB be the value of $NVEB$, let $NVEE$ be the <numeric value expression exponent>, and let VE be the value of $NVEE$.

Case:

- a) If at least one of VB and VE is the null value, then the result is the null value.
- b) If VB is 0 (zero) and VE is negative, then an exception condition is raised: *data exception — invalid argument for power function (2201F)*.
- c) If VB is 0 (zero) and VE is 0 (zero), then the result is 1 (one).
- d) If VB is 0 (zero) and VE is positive, then the result is 0 (zero).
- e) If VB is negative and VE is not equal to an exact numeric value with scale 0 (zero), then an exception condition is raised: *data exception — invalid argument for power function (2201F)*.
- f) If VB is negative and VE is equal to an exact numeric value with scale 0 (zero) that is an even number, then the result is the result of

$\text{EXP}(NVEE * \text{LN}(-NVEB))$

- g) If VB is negative and VE is equal to an exact numeric value with scale 0 (zero) that is an odd number, then the result is the result of

$-\text{EXP}(NVEE * \text{LN}(-NVEB))$

- h) Otherwise, the result is the result of

$\text{EXP}(\text{NVEE} * \text{LN}(\text{NVEB}))$

- 17) If <floor function> is specified, then let V be the result of the immediately contained <numeric value expression> NVE .

Case:

- a) If V is the null value, then the result is the null value.
b) Otherwise,

Case:

- i) If the most specific type of NVE is exact numeric, then the result is the greatest exact numeric value with scale 0 (zero) that is less than or equal to V . If this result is not representable by the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range (22003)*.
ii) Otherwise, the result is the greatest whole number that is less than or equal to V . If this result is not representable by the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range (22003)*.

- 18) If <ceiling function> is specified, then let V be the result of the immediately contained <numeric value expression> NVE .

Case:

- a) If V is the null value, then the result is the null value.
b) Otherwise,

Case:

- i) If the most specific type of NVE is exact numeric, then the result is the least exact numeric value with scale 0 (zero) that is greater than or equal to V . If this result is not representable by the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range (22003)*.
ii) Otherwise, the result is the least whole number that is greater than or equal to V . If this result is not representable by the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range (22003)*.

Conformance Rules

None.

19.7 <string value expression>

Function

Specify a character string value or a byte string value.

Format

```

<string value expression> ::= 
    <character string value expression>
  | <byte string value expression>

<character string value expression> ::= 
    <character string concatenation>
  | <character string factor>

<character string concatenation> ::= 
    <character string value expression> <concatenation operator> <character string factor>

<character string factor> ::= 
    <character string primary>

<character string primary> ::= 
    <value expression primary>
  | <string value function>

<byte string value expression> ::= 
    <byte string concatenation>
  | <byte string factor>

<byte string factor> ::= 
    <byte string primary>

<byte string primary> ::= 
    <value expression primary>
  | <string value function>

<byte string concatenation> ::= 
    <byte string value expression> <concatenation operator> <byte string factor>
  
```

Syntax Rules

« WG3:BER-094R1 »

- 1) The declared type of a <character string concatenation> is a character string type.
- 2) The declared type of a <character string factor> is the declared type of the simply contained <character string primary>.
- 3) The declared type of a <character string primary> shall be a character string type.
- 4) The declared type of a <byte string concatenation> is a byte string type.
- 5) The declared type of a <byte string factor> is the declared type of the simply contained <byte string primary>.
- 6) The declared type of a <byte string primary> shall be a byte string type.

General Rules

« WG3:BER-094R1 deleted nine GRs »

- 1) If the result of any <character string primary> simply contained in a <string value expression> is the null value, then the result of the <string value expression> is the null value.
- 2) If <character string concatenation> is specified, then:
 - a) In the remainder of this General Rule, the term “length” is taken to mean “length in characters”.
 - b) Let S_1 and S_2 be the result of the <character string value expression> and <character string factor>, respectively.

Case:

- i) If at least one of S_1 and S_2 is the null value, then the result of the <character string concatenation> is the null value.
- ii) Otherwise:
 - 1) Let S be the character string comprising S_1 followed by S_2 and let M be the length of S .
 - 2) S is replaced by

Case:

- A) If the <search condition> S_1 IS NORMALIZED AND S_2 IS NORMALIZED evaluates to True, then:

NORMALIZE (S)

- B) Otherwise, an implementation-defined (IA012) character string.
- 3) Let VL be the implementation-defined (IL013) maximum length of character strings.

Case:

- A) If M is less than or equal to VL , then the result of the <character string concatenation> is S with length M .
- B) If M is greater than VL and the right-most $M-VL$ characters of S are all the <whitespace> characters, then the result of the <character string concatenation> is the first VL characters of S with length VL .
- C) Otherwise, an exception condition is raised: *data exception — string data, right truncation (22001)*.

- 3) If <byte string concatenation> is specified, then let S_1 and S_2 be the result of the <byte string value expression> and <byte string factor>, respectively.

Case:

- a) If at least one of S_1 and S_2 is the null value, then the result of the <byte string concatenation> is the null value.
- b) Otherwise, let S be the byte string comprising S_1 followed by S_2 , let M be the length in bytes of S , and let VL be the implementation-defined (IL014) maximum length of byte strings.

Case:

- i) If M is less than or equal to VL , then the result of the <byte string concatenation> is S with length M .
- ii) If M is greater than VL and the right-most $M-VL$ bytes of S are all X'00', then the result of the <byte string concatenation> is the first VL bytes of S with length VL .
- iii) Otherwise, an exception condition is raised: *data exception — string data, right truncation (22001)*.

Conformance Rules

None.

19.8 <string value function>

Function

Specify a function yielding a character string value or a byte string value.

Format

```

<string value function> ::=

    <character string function>
  | <byte string function>

<character string function> ::=

    <substring function>
  | <fold>
  | <trim function>
  | <normalize function>

<substring function> ::=
  SUBSTRING <left paren> <character string value expression> <comma> <start position>
    [ <comma> <string length> ] <right paren>
  | LEFT <left paren> <character string value expression> <comma> <string length> <right
    paren>
  | RIGHT <left paren> <character string value expression> <comma> <string length> <right
    paren>

<fold> ::=
  { UPPER | toUpper | LOWER | toLower } <left paren> <character string value expression>
  <right paren>

<trim function> ::=
  TRIM <left paren> <trim source> [ <comma> <trim specification> [ <trim character string>
    ] ] <right paren>
  | lTrim <left paren> <trim source> <right paren>
  | rTrim <left paren> <trim source> <right paren>

<trim source> ::=
  <character string value expression>

<trim specification> ::=
  LEADING
  | TRAILING
  | BOTH

<trim character string> ::=
  <character string value expression>

<normalize function> ::=
  NORMALIZE <left paren> <character string value expression>
    [ <comma> <normal form> ] <right paren>

<normal form> ::=
  NFC
  | NFD
  | NFKC
  | NFKD

<byte string function> ::=
  <byte substring function>
  | <byte string trim function>

```

19.8 <string value function>

```

<byte substring function> ::= 
    SUBSTRING <left paren> <byte string value expression> <comma> <start position>
        [ <comma> <string length> ] <right paren>
    | LEFT <left paren> <byte string value expression> <comma> <string length> <right paren>
    | RIGHT <left paren> <byte string value expression> <comma> <string length> <right paren>

<byte string trim function> ::= 
    TRIM <left paren> <byte string trim source> [ <comma> <trim specification> [ <trim byte
        string> ] ] <right paren>
    | lTrim <left paren> <byte string trim source> <right paren>
    | rTrim <left paren> <byte string trim source> <right paren>

<byte string trim source> ::= 
    <byte string value expression>

<trim byte string> ::= 
    <byte string value expression>

<start position> ::= 
    <numeric value expression>

<string length> ::= 
    <numeric value expression>

```

**** Editor's Note (number 234) ****

SQL contains other possibly relevant string value functions such as:

- 1) OVERLAY for character strings.
See Language Opportunity [GQL-032](#).

**** Editor's Note (number 235) ****

SQL contains other string value functions that are probably not relevant such as:

- 1) CONVERT.
- 2) TRANSLITERATE.
- 3) CLASSIFIER.

Discussion is requires as to whether any of the functions named above should be incorporated into GQL.

See Language Opportunity [GQL-032](#).

**** Editor's Note (number 236) ****

Cypher contains other possibly relevant string value functions such as:

- 1) replace() - returns a string in which every occurrence of a specified string in the original string has been replaced by another (specified) string.
- 2) reverse() - returns a string in which the order of every character in the original string have been reversed.
- 3) type() - returns the string representation of the edge label. Note: that this would need modification for GQL that allows multiple labels on an edge that Cypher does not.
- 4) randomUUID() - returns a string value corresponding to a randomly generated UUID.

Discussion is requires as to whether any of the functions named above should be incorporated into GQL.

See Language Opportunity [GQL-032](#).

Syntax Rules

**** Editor's Note (number 237) ****

[BER-019] and [BER-094R1] established declared type propagation for the result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-291](#).

- 1) If <substring function> *CSF* is specified, then

Case:

- a) If LEFT is specified, then let *SRC* be <character string value expression>. *CSF* is equivalent to:

```
TRIM ( SRC , LEADING )
```

- b) If RIGHT is specified, then let *SRC* be <character string value expression>. *CSF* is equivalent to:

```
TRIM ( SRC , TRAILING )
```

- 2) If <trim function> is specified, then

Case:

- a) If lTrim is specified, then let *SRC* be <trim source>. <trim function> is equivalent to:

```
TRIM ( SRC , LEADING )
```

- b) If rTrim is specified, then let *SRC* be <trim source>. <trim function> is equivalent to:

```
TRIM ( SRC , TRAILING )
```

- 3) If <normalize function> is specified, then:

Case:

- a) If <normal form> is specified, then let *NF* be <normal form>.

- b) Otherwise, let *NF* be NFC.

- 4) If <byte substring function> *BSF* is specified, then

Case:

- a) If LEFT is specified, then let *BRC* be <byte string value expression>. *BSF* is equivalent to:

```
TRIM ( BRC , LEADING )
```

- b) If RIGHT is specified, then let *BRC* be <byte string value expression>. *BSF* is equivalent to:

```
TRIM ( BRC , TRAILING )
```

- 5) If <byte string trim function> is specified, then:

Case:

- a) If lTrim is specified, then let *SRC* be <byte string trim source>. <byte string trim function> is equivalent to:

```
TRIM ( SRC , LEADING )
```

- b) If rTrim is specified, then let *SRC* be <byte string trim source>. <byte string trim function> is equivalent to:

```
TRIM ( SRC , TRAILING )
```

General Rules

- 1) The declared type of <string value function> is the declared type of the immediately contained <character string function>, or <byte string function>.
- 2) The declared type of <character string function> is the declared type of the immediately contained <substring function>, <fold>, <trim function>, or <normalize function>.
- 3) if the most specific type of either a <start position> or a <string length> is not exact numeric with scale 0 (zero) then an exception condition is raised: *data exception — invalid value type (22G03)*.
- 4) The result of <string value function> is the result of the immediately contained <character string function> or <byte string function>.
- 5) If <substring function> *CSF* is specified, then the declared type of *CSF* is character string with maximum length equal to the maximum length of the <character string value expression>.
- 6) The result of <character string function> is the result of the immediately contained <substring function>, <fold>, <trim function>, or <normalize function>.
- 7) If <fold> is specified, then the declared type of the result of <fold> is that of the <character string value expression>.
- 8) If <trim function> is specified, then:
 - a) The declared type of the <trim function> is character string with maximum length equal to the maximum length of the <trim source>.
 - b) If a <trim character string> is specified, then if <trim character string> and <trim source> are not comparable then an exception condition is raised: *data exception — values not comparable (22G04)*.
- 9) The declared type of the <normalize function> is character string.
- 10) The declared type of <byte string function> is the declared type of the immediately contained <byte substring function>, or <byte string trim function>.
- 11) If <byte substring function> *BSF* is specified, then the declared type of *BSF* is the byte string type with maximum length equal to the maximum length of the <byte string value expression>.
- 12) The declared type of the <trim function> is the character string type with maximum length equal to the maximum length of the <trim source>.
- 13) If <substring function> is specified, then:
 - a) Let *C* be the result of the <character string value expression>, let *LC* be the length in characters of *C*, and let *S* be the result of the <start position>.
 - b) If <string length> is specified, then let *L* be the value of <string length> and let *E* be *S+L*; otherwise, let *E* be the larger of *LC + 1* (one) and *S*.
 - c) If at least one of *C*, *S*, and *L* is the null value, then the result of the <substring function> is the null value.

- d) If E is less than S , then an exception condition is raised: *data exception — substring error (22011)*.
- e) Case:
 - i) If S is greater than LC or if E is less than 1 (one), then the result of the <substring function> is the zero-length character string.
 - ii) Otherwise,
 - 1) Let $S1$ be the larger of S and 1 (one). Let $E1$ be the smaller of E and $LC+1$. Let $L1$ be $E1-S1$.
 - 2) The result of the <substring function> is a character string containing the $L1$ characters of C starting at character number $S1$ in the same order that the characters appear in C .
- 14) If <normalize function> is specified, then:
 - a) Let S be the result of <character string value expression>.
 - b) If S is the null value, then the result of the <normalize function> is the null value.
 - c) Let NR be S in the normalized form specified by NF in accordance with **Unicode Standard Annex #15**.
 - d) If the length in characters of NR is less than or equal to implementation-defined (IL013) maximum length of a character string, then the result of the <normalize function> is NR ; otherwise, an exception condition is raised: *data exception — string data, right truncation (22001)*.
- 15) If <fold> is specified, then:
 - a) Let S be the result of the <character string value expression>.
 - b) If S is the null value, then the result of the <fold> is the null value.
 - c) Case:
 - i) If **UPPER** is specified, then let FR be a copy of S in which every lower-case character that has a corresponding upper-case character or characters and every title case character that has a corresponding upper-case character or characters is replaced by that upper-case character or characters.
 - ii) If **LOWER** is specified, then let FR be a copy of S in which every upper-case character that has a corresponding lower-case character or characters and every title case character that has a corresponding lower-case character or characters is replaced by that lower-case character or characters.
 - d) FR is replaced by

Case:

 - i) If the <search condition> $s \text{ IS NORMALIZED}$ evaluated to *True*, then:
`NORMALIZE (FR)`
 - ii) Otherwise, FR .
 - e) Let FRL be the length in characters of FR and let $FRML$ be the implementation-defined (IL013) maximum length of a character string.
 - f) Case:

19.8 <string value function>

- i) If FRL is less than or equal to $FRML$, then the result of the <fold> is FR .
 - ii) Otherwise, the result of the <fold> is the first $FRML$ characters of FR . If any of the right-most ($FRL - FRML$) characters of FR are not <whitespace> characters, then a completion condition is raised: *data exception — string data, right truncation (22001)*.
- 16) If <trim function> is specified, then:
- a) Let S be the result of the <trim source>.
 - b) If <trim character string> is specified, then let SC be the value of <trim character string>; otherwise, let SC be the zero-length character string.
 - c) If at least one of S and SC is the null value, then the result of the <trim function> is the null value.
 - d) If the length in characters of SC is not 0 (zero) or 1 (one), then an exception condition is raised: *data exception — trim error (22027)*.
 - e) Case:
 - i) If BOTH is specified or if no <trim specification> is specified, then the result of the <trim function> is the value of S with
 - Case:
 - 1) If the length in characters of SC is 1 (one), then any leading or trailing characters equal to SC removed.
 - 2) Otherwise, any leading or trailing <whitespace> characters removed.
 - ii) If TRAILING is specified, then the result of the <trim function> is the value of S with
 - Case:
 - 1) If the length in characters of SC is 1 (one), then any trailing characters equal to SC removed.
 - 2) Otherwise, any trailing <whitespace> characters removed.
 - iii) If LEADING is specified, then the result of the <trim function> is the value of S with
 - Case:
 - 1) If the length in characters of SC is 1 (one), then any leading characters equal to SC removed.
 - 2) Otherwise, any leading <whitespace> characters removed.
- 17) The result of <byte string function> is the result of the immediately contained <byte substring function>, or <byte string trim function>.
- 18) If <byte substring function> is specified, then:
- a) Let B be the result of the <byte string value expression>, let LB be the length in bytes of B , and let S be the value of the <start position>.
 - b) If <string length> is specified, then let L be the value of <string length> and let E be $S+L$; otherwise, let E be the larger of $LB+1$ and S .
 - c) If at least one of B , S , and L is the null value, then the result of the <byte substring function> is the null value.

- d) If E is less than S , then an exception condition is raised: *data exception — substring error (22011)*.
 - e) Case:
 - i) If S is greater than LB or if E is less than 1 (one), then the result of the <byte substring function> is the zero-length byte string.
 - ii) Otherwise:
 - 1) Let $S1$ be the larger of S and 1 (one). Let $E1$ be the smaller of E and $LB+1$. Let $L1$ be $E1-S1$.
 - 2) The result of the <byte substring function> is a byte string containing $L1$ bytes of B starting at byte number $S1$ in the same order that the bytes appear in B .
- 19) If <byte string trim function> is specified, then:
- a) Let S be the result of the <byte string trim source>.
 - b) Let SO be the value of <trim byte string>.
 - c) If at least one of S and SO is the null value, then the result of the <byte string trim function> is the null value.
 - d) If the length in bytes of SO is not 1 (one), then an exception condition is raised: *data exception — trim error (22027)*.
 - e) Case:
 - i) If BOTH is specified or if no <trim specification> is specified, then the result of the <byte string trim function> is the value of S with any leading or trailing bytes equal to SO removed.
 - ii) If TRAILING is specified, then the result of the <byte string trim function> is the value of S with any trailing bytes equal to SO removed.
 - iii) If LEADING is specified, then the result of the <byte string trim function> is the value of S with any leading bytes equal to SO removed.

Conformance Rules

None.

19.9 <datetime value expression>

Function

Specify a datetime value.

Format

```
<datetime value expression> ::=  
  <datetime term>  
  | <duration value expression> <plus sign> <datetime term>  
  | <datetime value expression> <plus sign> <duration term>  
  | <datetime value expression> <minus sign> <duration term>  
  
<datetime term> ::=  
  <datetime factor>  
  
<datetime factor> ::=  
  <datetime primary>  
  
<datetime primary> ::=  
  <value expression primary>  
  | <datetime value function>
```

Syntax Rules

**** Editor's Note (number 238) ****

[BER-019] and [BER-094R1] established declared type propagation for the result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-292](#).

None.

General Rules

- 1) If the most specific type of a <datetime primary> is not datetime then an exception condition is raised: *data exception — invalid value type (22G03)*.
- 2) If the result of any <datetime primary>, <duration value expression>, <datetime value expression>, or <duration term> simply contained in a <datetime value expression> is the null value, then the result of the <datetime value expression> is the null value.
- 3) The result of a <datetime primary> is the result of the immediately contained <value expression primary> or <datetime value function>.
- 4) If a <datetime value expression> immediately contains the operator <plus sign> or <minus sign>, then the result is evaluated as specified in clause 14 “Date and time arithmetic” of [ISO 8601-2:2019](#).

Conformance Rules

None.

19.10 <datetime value function>

Function

Specify a function yielding a value of type datetime.

Format

```
<datetime value function> ::=  
  <date function>  
  | <time function>  
  | <datetime function>  
  | <local time function>  
  | <local datetime function>  
  
<date function> ::=  
  CURRENT_DATE  
  | DATE <left paren> [ <date function parameters> ] <right paren>  
  
<time function> ::=  
  CURRENT_TIME  
  | TIME <left paren> [ <time function parameters> ] <right paren>  
  
<local time function> ::=  
  LOCALTIME  
  | LOCALTIME <left paren> [ <time function parameters> ] <right paren>  
  
<datetime function> ::=  
  CURRENT_TIMESTAMP  
  | DATETIME <left paren> [ <datetime function parameters> ] <right paren>  
  
<local datetime function> ::=  
  LOCALTIMESTAMP  
  | LOCALDATETIME <left paren> [ <datetime function parameters> ] <right paren>  
  
<date function parameters> ::=  
  <date string>  
  | <mmap value constructor>  
  
<time function parameters> ::=  
  <time string>  
  | <mmap value constructor>  
  
<datetime function parameters> ::=  
  <datetime string>  
  | <mmap value constructor>
```

** Editor's Note (number 239) **

Cypher has a raft of datetime functions, some of which may not be already incorporated into GQL. These should be checked for suitability of inclusion. See [Language Opportunity GQL-196](#).

Syntax Rules

** Editor's Note (number 240) **

[BER-019] and [BER-094R1] established declared type propagation for the result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-293](#).

- 1) CURRENT_DATE is equivalent to:

```
date()
```

- 2) CURRENT_TIME is equivalent to:

```
time()
```

- 3) LOCALTIME is equivalent to:

```
localtime()
```

- 4) CURRENT_TIMESTAMP is equivalent to:

```
datetime()
```

- 5) LOCALTIMESTAMP is equivalent to:

```
localdatetime()
```

General Rules

- 1) The declared type of a <date function> is DATE. The declared type of a <time function> is TIME. The declared type of a <datetime function> is DATETIME. The declared type of a <local time function> is LOCALTIME. The declared type of a <local datetime function> is LOCALDATETIME.
- 2) If the set of <map key>s contained in a <date function parameters> is not one of:
 - a) 'year'
 - b) 'year' and 'month'
 - c) 'year', 'month', and 'day'
 - d) 'year' and 'week'
 - e) 'year', 'week', and 'dayOfWeek'
 - f) 'year' and 'ordinalDay'
 then an exception is raised: *data exception — invalid datetime function map key (22G05)*.
- 3) If the set of <map key>s contained in a <time function parameters> is not one of:
 - a) 'hour'
 - b) 'hour' and 'minute'
 - c) 'hour', 'minute', and 'second'
 - d) 'hour', 'minute', and 'second'
 - e) 'hour', 'minute', and 'second'
 - f) 'hour', 'minute', 'second' and one or more of 'millisecond', 'microsecond', and 'nanosecond'.
 - g) In addition, if <time function parameters> is contained in a <time function>, then, the set of permitted <map key>s also includes 'timezone'.

then an exception is raised: *data exception — invalid datetime function map key (22G05)*.

- 4) If the set of <map key>s contained in a <datetime function parameters> is not one of:
 - a) 'year', 'month', 'day', and any of the set of <map key>s permitted in a <time function parameters>.
 - b) 'year', 'week', 'dayOfWeek', and any of the set of <map key>s permitted in a <time function parameters>.
 - c) 'year', 'ordinalDay', and any of the set of <map key>s permitted in a <time function parameters>.
- then an exception is raised: *data exception — invalid datetime function map key (22G05)*.
- 5) If a <datetime function parameters> DFP is contained in <local datetime function> and DFP contains the <map key> 'timezone' then an exception is raised: *data exception — invalid datetime function map key (22G05)*.
 - 6) If the value of each <map value> associated with the <map key> 'millisecond' is not between 0 (zero) and 999, or the value associated with 'microsecond', not between 0 (zero) and 999999 and the value for 'nanosecond' not between 0 (zero) and 999999999, or, if more than one of 'millisecond', 'microsecond', and 'nanosecond' is specified, any value exceeds 999 then an exception is raised: *data exception — invalid datetime function map value (22G06)*. If the values associated with other <map key>s do not conform to the range of values specified in clause 4.3 "Time scale components and units" of ISO 8601-1:2019 with, for the 'timezone' key, the optional addition of an IANA Time Zone Database time zone designator enclosed between a <left bracket> and a <right bracket> or, if both an IANA Time Zone Database time zone designator and a ISO 8601-1:2019 time shift specification are present in a 'timezone' key and the IANA Time Zone Database time zone designator does not resolve to the same value as the ISO 8601-1:2019 time shift specification then an exception is raised: *data exception — invalid datetime function map value (22G06)*.

**** Editor's Note (number 241) ****

During the discussion of WG3:W08-014 (See WG3:W09-001) it was apparent that there was no consensus on the best way to specify a time zone. Should the standard allow both an explicit offset and an IANA time zone designator and should the combination also be allowed. This topic need to be revisited to establish consensus. See Possible Problem [GQL-182](#).

- 7) If <date function parameters>, <time function parameters>, or <datetime function parameters> are specified, the map is transformed, respectively into an equivalent <date string>, <time string>, or <datetime string>.
- 8) Case:
 - a) If the <datetime value function>s date(), time(), datetime(), localtime(), and localdatetime() have no parameters then they respectively return the current date, time, datetime, localtime and localdatetime.

For time() the returned value has a displacement equal to the current time zone displacement of the time zone identified by the current time zone identifier of the session context.

For datetime() the returned value has a displacement equal to the current time zone displacement of the time zone identified by the current time zone identifier of the session context and a time zone identifier equal to the current time zone identifier of the session context.

 - b) Otherwise, they return respectively the date, time, datetime, localtime and localdatetime values associated with the representation as defined by ISO 8601-1:2019 and ISO 8601-2:2019.

Conformance Rules

None.

19.11 <duration value expression>

Function

Specify a duration value.

Format

```

<duration value expression> ::=

  <duration term>
  | <duration value expression 1> <plus sign> <duration term 1>
  | <duration value expression 1> <minus sign> <duration term 1>
  | <left paren> <datetime value expression> <minus sign> <datetime term> <right paren>

<duration term> ::=

  <duration factor>
  | <duration term 2> <asterisk> <factor>
  | <duration term 2> <solidus> <factor>
  | <term> <asterisk> <duration factor>

<duration factor> ::=

  [ <sign> ] <duration primary>

<duration primary> ::=

  <value expression primary>
  | <duration value function>

<duration value expression 1> ::=
  <duration value expression>

<duration term 1> ::=
  <duration term>

<duration term 2> ::=
  <duration term>

```

Syntax Rules

**** Editor's Note (number 242) ****

[BER-019] and [BER-094R1] established declared type propagation for the result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-294](#).

- 1) If <duration term>, *DT* immediately contains <solidus>, then let *DT2* be the <duration term 2> immediately contained in *DT* and *F* be the <factor> immediately contained in *DT*. *DT* is effectively replaced by:

*DT2 * (1 / F)*

General Rules

- 1) The declared type of a <duration value expression> is duration.
- 2) If the most specific type of a <value expression primary> immediately contained in a <duration primary> is not duration then an exception condition is raised: *data exception — invalid value type (22G03)*.

- 3) If <datetime value expression> is specified, then if <datetime value expression> and <datetime term> are not comparable then an exception condition is raised: *data exception — values not comparable (22G04)*.
- 4) If <duration term> immediately contains <asterisk>, then let V be the result of the immediately contained <term> or <factor>. The result of the <duration term> is the duration specified by clause 14.3, “Multiplication” of [ISO 8601-2:2019](#) with V as the coefficient, and <duration term 2> or <duration factor> as durationA.
- 5) The result of <duration value expression> DVE is

Case:

- a) If DVE immediately contains <duration term> then the value of DVE is the result of <duration term>.
- b) If DVE immediately contains <duration value expression 1> then the result is as specified by clause 14, “Date and time arithmetic” of [ISO 8601-2:2019](#).
- c) Otherwise, the value of DVE is the duration that if added, as specified by clause 14, “Date and time arithmetic” of [ISO 8601-2:2019](#), to the <datetime term> would yield the same value as <datetime value expression>.

Conformance Rules

None.

19.12 <duration value function>

Function

Specify a function yielding a value of type duration.

Format

```
<duration value function> ::=  

    <duration function>  

  | <duration absolute value function>  
  

<duration function> ::=  

  DURATION <left paren> <duration function parameters> <right paren>  
  

<duration function parameters> ::=  

    <duration string>  

  | <map value constructor>  
  

<duration absolute value function> ::=  

  ABS <left paren> <duration value expression> <right paren>
```

**** Editor's Note (number 243) ****

Cypher has a raft of duration functions, some of which may not be already incorporated into GQL. These should be checked for suitability of inclusion. See [Language Opportunity GQL-197](#).

Syntax Rules

**** Editor's Note (number 244) ****

[BER-019] and [BER-094R1] established declared type propagation for the result. This Subclause does not do that yet and requires corresponding Syntax Rules. See [Possible Problem GQL-295](#).

None.

General Rules

- 1) The declared type of a <duration function> is DURATION.
- 2) If a <map key>s contained in a <duration function parameters> is not one of 'years', 'months', 'days', 'weeks', 'days', 'hours', 'minutes', 'seconds', 'milliseconds', 'microseconds', or 'nanoseconds' then an exception condition is raised: *data exception — invalid duration function map key (22G07)*.
- 3) If <duration function parameters> is specified, the map is transformed into an equivalent <duration string>.
- 4) The result of a <duration function> is the duration specified for the <duration string> defined in clause 5.5.2 "Duration" of [ISO 8601-1:2019](#) as extended by clauses 4.3 "Additional explicit forms" and 4.4 "Numerical extensions" of [ISO 8601-2:2019](#).
- 5) If <duration absolute value function> is specified, then let N be the result of the <duration value expression>.

Case:

- a) If N is the null value, then the result is the null value.
- b) If $N \geq 0$ (zero), then the result is N .
- c) Otherwise, the result is $-1 * N$.

Conformance Rules

None.

« WG3:BER-094R1 deleted three Subclause »

19.13 <list value expression>

« WG3:BER-094R1 deleted one Editor's Note »

Function

Specify a list value.

Format

```
<list value expression> ::=  
    <list concatenation>  
  | <list primary>  
  
<list concatenation> ::=  
    <list value expression 1> <concatenation operator> <list primary>  
  
<list value expression 1> ::=  
    <list value expression>  
  
<list primary> ::=  
    <list value function>  
  | <value expression primary>
```

Syntax Rules

« WG3:BER-094R1 »

- 1) The declared type of a <list primary> shall be a list type.
- 2) If <list concatenation> is specified, then:
 - a) The General Rules of Subclause 21.11, “Result of value type combinations”, are applied with the most specific types of <list value expression 1> and <list primary> as *DTSET*; let *DT* be the *RESTYPE* returned from the application of those General Rules.
 - b) Let *IDMC* be the implementation-defined (IL015) maximum cardinality of a list type.
 - c) The declared type of the result of <list concatenation> is *DT*.

General Rules

« WG3:BER-094R1 deleted three GRs »

« WG3:BER-094R1 »

- 1) If <list concatenation> is specified, then let *LV1* be the result of <list value expression 1> and let *LV2* be the value of <list primary>.Case:
 - a) If at least one of *LV1* and *LV2* is the null value, then the result of the <list concatenation> is the null value.
 - b) If the sum of the cardinality of *LV1* and the cardinality of *LV2* is greater than *IDMC*, then an exception condition is raised: *data exception — list data, right truncation (22G0B)*.

- c) Otherwise, the result is the list comprising every element of LV1 followed by every element of LV2.

Conformance Rules

None.

19.14 <list value function>

« WG3:BER-094R1 deleted one Editor's Note »

Function

Specify a function yielding a value of a list type.

Format

```
« WG3:BER-094R1 »

<list value function> ::= 
  <trim list function>
« WG3:BER-094R1 deleted one production »

<trim list function> ::= 
  TRIM <left paren> <list value expression> <comma> <numeric value expression> <right paren>
« WG3:BER-094R1 deleted two Editor's Notes »
```

Syntax Rules

« WG3:BER-094R1 »

- 1) If <trim list function> is specified, then:
 - a) The declared type of the <numeric value expression> shall be an exact numeric type with scale 0 (zero).
 - b) The declared type of the <trim list function> is the declared type of the immediately contained <list value expression>.

General Rules

« WG3:BER-094R1 deleted 5 GRs »

- 1) The result of <trim list function> is defined as follows:
 - a) Let NV be the result of the <numeric value expression>.
 - b) If NV is the null value, then the result is the null value and no further General Rules of this Subclause are applied.
 - c) If NV is less than 0 (zero), then an exception condition is raised: *data exception — list element error (22G0C)*.
 - d) Let AV be the result of the <list value expression>.
 - e) If AV is the null value, then the result is the null value and no further General Rules of this Subclause are applied.
 - f) Let AC be the cardinality of AV .
 - g) If NV is greater than AC , then an exception condition is raised: *data exception — list element error (22G0C)*.
 - h) Let N be $AC - NV$.

i) Case:

- i) If $N = 0$ (zero), then the result is a list whose cardinality is 0 (zero).
- ii) Otherwise, the result is a list of N elements such that for every i , $1 \leq i \leq N$, the value of the i -th element of the result is the value of the i -th element of AV .

Conformance Rules

None.

19.15 <list value constructor>

« WG3:BER-094R1 deleted one Editor's Note »

Function

Specify construction of a list.

Format

```
<list value constructor> ::=  
  <list value constructor by enumeration>  
  
<list value constructor by enumeration> ::=  
  <list value type name> <left bracket> <list element list> <right bracket>  
  
<list element list> ::=  
  <list element> [ { <comma> <list element> }... ]  
  
<list element> ::=  
  <value expression>
```

Syntax Rules

« WG3:BER-094R1 »

- 1) The declared type of <list value constructor> is the declared type of the immediately contained <list value constructor by enumeration>.
- 2) The General Rules of Subclause 21.11, “Result of value type combinations”, are applied with the declared types of the <list element>s immediately contained in the <list element list> of the <list value constructor by enumeration> as *DTSET*; let *DT* be the *RESTYPE* returned from the application of those General Rules. The declared type of the <list value constructor by enumeration> is a list type with element type *DT*.
- 3) The number of <list element>s in the <list element list> of the <list value constructor by enumeration> shall be less or equal than the implementation-defined (IL015) maximum cardinality for list types whose element type is *DT*.

General Rules

« WG3:BER-094R1 deleted three GRs »

- 1) The result of <list value constructor by enumeration> is a list whose *i*-th element is the value of the *i*-th <list element> immediately contained in the <list element list>, cast as the value type of *DT*.

Conformance Rules

None.

« WG3:BER-094R1 deleted four Subclauses »

« WG3:BER-038R1 deleted one Subclause »

19.16 <map value constructor>

** Editor's Note (number 245) **

WG3:W05-020 suggests that discussion of this Subclause is needed to establish consensus. See Possible Problem [GQL-083](#).

Function

Specify construction of a map.

Format

```

<map value constructor> ::= 
    <map value constructor by enumeration>

<map value constructor by enumeration> ::= 
    MAP <left brace> <map element list> <right brace>

<map element list> ::= 
    <map element> [ { <comma> <map element> }... ]

<map element> ::= 
    <map key> <map value>

<map key> ::= 
    <value expression> <colon>

<map value> ::= 
    <value expression>

```

Syntax Rules

** Editor's Note (number 246) **

[BER-019] and [BER-094R1] established declared type propagation for the result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-296](#).

None.

General Rules

- 1) The declared type of <map value constructor> is the declared type of the immediately contained <map value constructor by enumeration>.
- 2) If the declared type of a <value expression> immediately contained in <map key> is not a <predefined type> then an exception condition is raised: *data exception — invalid value type (22G03)*.
- 3) The General Rules of Subclause 21.11, “Result of value type combinations”, are applied with the declared types of the <map key>s immediately contained in the <map element list> of the <map value constructor by enumeration> as *DTSET*; let *KT* be the *RESTYPE* returned from the application of those General Rules.
- 4) The General Rules of Subclause 21.11, “Result of value type combinations”, are applied with the declared types of the <map value>s immediately contained in the <map element list> of the <map value constructor by enumeration> as *DTSET*; let *DT* be the *RESTYPE* returned from the application of those General Rules.

- 5) The declared type of the <map value constructor by enumeration> is a map type with key element type KT and value element type DT .
- 6) The value of <map value constructor> is the value of the immediately contained <map value constructor by enumeration>.
- 7) The result of <map value constructor by enumeration> is a map whose elements are the values of the <map element>s immediately contained in the <map element list>.
- 8) The value of <map element> is a (key, value) pair where the value of the “key” is the value of the <map key>s immediately contained in the <map element>, cast as the value type of KT and the value of the “value” is the value of the <map value> immediately contained in the <map element>, cast as the value type of DT .

Conformance Rules

None.

19.17 <record value constructor>

« WG3:BER-040R3 deleted one Editor's Note »

Function

Specify construction of a record.

Format

```
« WG3:BER-040R3 »

<record value constructor> ::=  
  [ RECORD ] <field specification>  
  
<field specification> ::=  
  <left brace> [ <field list> ] <right brace>  
« WG3:BER-040R3 deleted one production »  
  
<field list> ::=  
  <field> [ { <comma> <field> }... ]  
« WG3:BER-040R3 deleted two productions »
```

Syntax Rules

« WG3:BER-040R3 deleted two SRs »

- 1) Let *RVC* be the <record value constructor>.
- 2) Every <field specification> specifies the fields specified by the immediately contained <field list>.
- 3) Every <field list> *FL* specifies the fields defined by the <field>s simply contained in *FL*.
- 4) Let *FS* be the <field specification> immediately contained in *RVC*, let *n* be the number of fields specified by *FS*. For every $i \in \{1, \dots, n\}$, let *F_i* be the *i*-th field specified by *FS*, and let *FT_i* be the field type of *F_i*.
- 5) Let *RT* be determined as follows:
 - a) Let *RT₀* be determined as follows:
 - i) If *n* is 0 (zero), then *RT₀* is


```
RECORD {}
```
 - ii) Otherwise, *RT₀* is


```
RECORD { FT1, ..., FTn }
```
 - b) *RT* is the <record type> obtained by the application of Syntax Rules for <record type> to *RT₀*.
- 6) The declared type of *RVC* is the record type specified by *RT*.

General Rules

« WG3:BER-040R3 deleted six GRs »

« WG3:BER-040R3 »

- 1) The General Rules of <record type> are applied to *RT*.
- 2) The value of *RVC* is the record whose set of fields are the fields defined by the <field>s specified by *FS*.

Conformance Rules

None.

« WG3:BER-040R3 »

19.18 <field>

Function

Specify a field.

Format

```
<field> ::=  
  <field name> <colon> <value expression>
```

Syntax Rules

- 1) Let FL be the <field list> that simply contains a <field> F .
- 2) The <field name> shall not be equivalent to the <field name> of any other <field> simply contained in FL .
- 3) The <field name> specifies the *field name* of the field defined by F .
- 4) The <value expression> specifies the *field value expression* of F .
- 5) The field type of F is the pair comprising the field name of the field defined by F and the declared type of the field value expression of F .

General Rules

- 1) The *field value* of the field defined by F is the value of the field value expression of F .

Conformance Rules

- 1) Without Feature GD01, “Nested record types”, conforming language shall not contain a <field> that simply contains a <record value constructor>.

19.19 <property reference>

Function

Reference a property of a graph element.

Format

« Consequence of WG3:BER-094R1 »

```
<property reference> ::=  
  <reference value expression> <period> <property name>
```

** Editor's Note (number 247) **

<property reference> was not defined in SQL/PGQ by WG3:ERF-034. This is an editorial suggestion that serves as a place holder and requires further refinement. WG3:W05-020 also noted that discussion was needed to establish consensus. See Possible Problem [GQL-085](#).

Syntax Rules

** Editor's Note (number 248) **

[BER-019] and [BER-094R1] established declared type propagation for the result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-297](#).

None.

General Rules

- 1) Let PR be the <property reference>.
- 2) Let GE be the result of the <reference value expression>. If GE is not a graph element, raise an exception condition.

** Editor's Note (number 249) **

Details of static and dynamic type checking to be determined. See Possible Problem [GQL-007](#).

- 3) If GE has a property P whose property name is the <property name>, then the result of PR is the property value of P ; otherwise, the result of PR is the null value.

Conformance Rules

« WG3:BER-030 »

- 1) Without Feature G090, “Property reference”, conforming SQL language shall not contain a <property reference>.

19.20 <value query expression>

** Editor's Note (number 250) **

The semantics of this Subclause have not been defined. The appropriate Syntax and General Rules must be defined. See Possible Problem [GQL-202](#).

Function

Define a <value query expression>.

Format

```
<value query expression> ::=  
    VALUE <nested query specification>
```

Syntax Rules

** Editor's Note (number 251) **

[BER-019] and [BER-094R1] established declared type propagation for the result. This Subclause does not do that yet and requires corresponding Syntax Rules. See Possible Problem [GQL-278](#). See Possible Problem [GQL-298](#).

None.

General Rules

None.

Conformance Rules

None.

19.21 <case expression>

Function

Specify a conditional value.

Format

```
<case expression> ::=  
  <case abbreviation>  
 | <case specification>  
  
<case abbreviation> ::=  
  NULLIF <left paren> <value expression> <comma> <value expression> <right paren>  
 | COALESCE <left paren> <value expression>  
   { <comma> <value expression> }... <right paren>  
  
<case specification> ::=  
  <simple case>  
 | <searched case>  
  
<simple case> ::=  
  CASE <case operand> <simple when clause>... [ <else clause> ] END  
  
<searched case> ::=  
  CASE <searched when clause>... [ <else clause> ] END  
  
<simple when clause> ::=  
  WHEN <when operand list> THEN <result>  
  
<searched when clause> ::=  
  WHEN <search condition> THEN <result>  
  
<else clause> ::=  
  ELSE <result>  
  
<case operand> ::=  
  <non-parenthesized value expression primary>  
 | <element reference>  
  
<when operand list> ::=  
  <when operand> [ { <comma> <when operand> }... ]  
  
<when operand> ::=  
  <non-parenthesized value expression primary>  
 | <comparison predicate part 2>  
 | <null predicate part 2>  
 | <directed predicate part 2>  
 | <labeled predicate part 2>  
 | <source predicate part 2>  
 | <destination predicate part 2>  
  
<result> ::=  
  <result expression>  
 | NULL  
  
<result expression> ::=  
  <value expression>
```

Syntax Rules

- 1) If a <case expression> specifies a <case abbreviation>, then:
 - a) NULLIF (v_1, v_2) is equivalent to the following <case specification>:


```
CASE WHEN
    V1=V2 THEN
    NULL ELSE V1
END
```
 - b) COALESCE (v_1, v_2) is equivalent to the following <case specification>:


```
CASE
    WHEN NOT V1 IS NULL THEN V1
    ELSE V2
END
```
 - c) COALESCE (v_1, v_2, \dots, v_n), for $n \geq 3$, is equivalent to the following <case specification>:


```
CASE
    WHEN NOT V1 IS NULL THEN V1
    ELSE COALESCE (V2, \dots, Vn)
END
```
- 2) If a <case specification> specifies a <simple case>, then let CO be the <case operand>.
 - a) If any <when operand> is <directed predicate part 2>, <labeled predicate part 2>, <source predicate part 2> or <destination predicate part 2>, then CO shall be <element reference> and every <when operand> shall be <directed predicate part 2>, <labeled predicate part 2>, <source predicate part 2> or <destination predicate part 2>; otherwise, CO shall not be <element reference> and no <when operand> shall be <directed predicate part 2>, <labeled predicate part 2>, <source predicate part 2> or <destination predicate part 2>.
 - b) Let N be the number of <simple when clause>s.
 - c) For each i between 1 (one) and N , let WOL_i be the <when operand list> of the i -th <simple when clause>. Let $M(i)$ be the number of <when operand>s simply contained in WOL_i . For each j between 1 (one) and $M(i)$, let $WO_{i,j}$ be the j -th <when operand> simply contained in WOL_i .
 - d) For each i between 1 (one) and N , and for each j between 1 (one) and $M(i)$,

Case:

 - i) If $WO_{i,j}$ is a <non-parenthesized value expression primary>, then let $EWO_{i,j}$ be:
$$= WO_{i,j}$$
 - ii) Otherwise, let $EWO_{i,j}$ be $WO_{i,j}$.
 - e) Let R_i be the <result> of the i -th <simple when clause>.
 - f) If <else clause> is specified, then let $CEEC$ be that <else clause>; otherwise, let $CEEC$ be the zero-length character string.
 - g) The <simple case> is equivalent to a <searched case> in which the i -th <searched when clause> takes the form:

```
WHEN ( CO EWOi,1 ) OR
... OR
( CO EWOi,M(i) )
THEN Ri
```

- h) The <else clause> of the equivalent <searched case> takes the form:

CEEC

- i) The Conformance Rules of the Subclauses of Clause 18, "Predicates", are applied to the result of this syntactic transformation.

NOTE 144 — The specific Subclauses of Clause 18, "Predicates", are determined by the predicates that are created as a result of the syntactic transformation.

- 3) At least one <result> in a <case specification> shall specify a <result expression>.

- 4) If an <else clause> is not specified, then ELSE NULL is implicit.

« WG3:BER-094R1 »

- 5) The General Rules of Subclause 21.11, "Result of value type combinations", are applied with the set of declared types of all <result expression>s in the <case specification> as DTSET; let DT be the RESTYPE returned from the application of those General Rules. The declared type of the <case specification> is RT.

General Rules

« WG3:BER-094R1 deleted one GR »

- 1) Case:

- a) If a <result> specifies NULL, then its value is the null value.
- b) If a <result> specifies a <value expression>, then its value is the result of that <value expression>.

- 2) Case:

- a) If the result of the <search condition> of some <searched when clause> in a <case specification> is *True*, then the result of the <case expression> is the result of the <result> of the first (left-most) <searched when clause> whose <search condition> evaluates to *True*, cast as the declared type of the <case specification>.
- b) If no <search condition> in a <case specification> evaluates to *True*, then the result of the <case expression> is the result of the <result> of the explicit or implicit <else clause>, cast as the declared type of the <case specification>.

19.22 <cast specification>

Function

Specify a data conversion.

Format

```

<cast specification> ::==
  CAST <left paren> <cast operand> AS <cast target> <right paren>

<cast operand> ::==
  <value expression>
  | <null literal>

<cast target> ::==
  <predefined type>

```

Syntax Rules

- 1) Let *TD* be the data type identified by <predefined type>.
- 2) The declared type of the result of the <cast specification> is *TD*.
- 3) If the <cast operand> is a <value expression>, then let *SD* be the declared type of the <value expression>.
- 4) If the <cast operand> is a <value expression>, then the valid combinations of *TD* and *SD* in a <cast specification> are given by the following table. "Y" indicates that the combination is syntactically valid without restriction; "M" indicates that the combination is valid subject to other Syntax Rules in this Subclause being satisfied; and "N" indicates that the combination is not valid.

<i>SD</i>	EN	UN	AN	C	D	T	DT	DU	BO	B	<i>TD</i>
EN	Y	Y	Y	Y	N	N	N	N	N	N	
UN	Y	Y	Y	Y	N	N	N	N	N	N	
AN	Y	Y	Y	Y	N	N	N	N	N	N	
C	Y	Y	Y	Y	Y	Y	Y	Y	Y	N	
D	N	N	N	Y	Y	N	Y	N	N	N	
T	N	N	N	Y	N	Y	Y	N	N	N	
DT	N	N	N	Y	Y	Y	Y	N	N	N	
DU	N	N	N	Y	N	N	N	Y	N	N	
BO	N	N	N	Y	N	N	N	N	Y	N	
B	N	N	N	N	N	N	N	N	N	Y	

Where:

EN = Signed Exact Numeric
 UN = Unsigned Exact Numeric
 AN = Approximate Numeric
 C = Character String
 D = Date
 T = Time & Localtime
 DT = Datetime & Localdatetime
 DU = Duration
 BO = Boolean
 B = Byte String

General Rules

- 1) Let CS be the <cast specification>. If the <cast operand> is a <value expression> VE , then let SV be the result of VE .
- 2) Case:
 - a) If the <cast operand> specifies NULL, then the result of CS is the null value and no further General Rules of this Subclause are applied.
 - b) If SV is the null value, then the result of CS is the null value and no further General Rules of this Subclause are applied.
- 3) If TD is a signed exact numeric type, then
 - Case:
 - a) If SD is a numeric type, then
 - Case:
 - i) If there is a representation of SV in the value type TD that does not lose any leading significant digits after rounding or truncating if necessary, then TV is that representation. The choice of whether to round or truncate is implementation-defined (IA005).
 - ii) Otherwise, an exception condition is raised: *data exception — numeric value out of range (22003)*.
 - b) If SD is character string type, then SV is replaced by SV with any leading or trailing <whitespace> removed.
 - Case:
 - i) If SV does not comprise a <signed numeric literal> as defined by the rules for <literal> in Subclause 20.1, “<literal>”, then an exception condition is raised: *data exception — invalid character value for cast (22018)*.
 - ii) Otherwise, let LT be that <signed numeric literal>. The <cast specification> is equivalent to `CAST (LT AS TD)`.
 - 4) If TD is an unsigned exact numeric type, then
 - Case:
 - a) If SD is a numeric type, then
 - Case:
 - i) If there is a representation of SV in the value type TD that does not lose any leading significant digits or the sign after rounding or truncating if necessary, then TV is that representation. The choice of whether to round or truncate is implementation-defined (IA005).
 - ii) Otherwise, an exception condition is raised: *data exception — numeric value out of range (22003)*.
 - b) If SD is character string type, then SV is replaced by SV with any leading or trailing <whitespace> removed.
 - Case:

- i) If *SV* does not comprise an <unsigned numeric literal> as defined by the rules for <literal> in Subclause 20.1, “<literal>”, then an exception condition is raised: *data exception — invalid character value for cast (22018)*.
 - ii) Otherwise, let *LT* be that <signed numeric literal>. The <cast specification> is equivalent to *CAST (LT AS TD)*.
- 5) If *TD* is an approximate numeric type, then
- Case:
- a) If *SD* is a numeric type, then
- Case:
- i) If there is a representation of *SV* in the value type *TD* that does not lose any leading significant digits after rounding or truncating if necessary, then *TV* is that representation. The choice of whether to round or truncate is implementation-defined (IA005).
 - ii) Otherwise, an exception condition is raised: *data exception — numeric value out of range (22003)*.
- b) If *SD* is a character string type, then *SV* is replaced by *SV* with any leading or trailing <whitespace> removed.
- Case:
- i) If *SV* does not comprise a <signed numeric literal> as defined by the rules for <literal> in Subclause 20.1, “<literal>”, then an exception condition is raised: *data exception — invalid character value for cast (22018)*.
 - ii) Otherwise, let *LT* be that <signed numeric literal>. The <cast specification> is equivalent to *CAST (LT AS TD)*.
- 6) If *TD* is a fixed-length character string type, then let *LTD* be the length in characters of *TD*.
- Case:
- a) If *SD* is an exact numeric type, then:
 - i) Let *YP* be the shortest character string that conforms to the definition of <exact numeric literal> in Subclause 20.1, “<literal>”, whose scale is the same as the scale of *SD*, whose interpreted value is the absolute value of *SV*, and that is not an <unsigned hexadecimal integer>.
 - ii) Case:
 - 1) If *SV* is less than 0 (zero), then let *Y* be the result of ‘-’ || *YP*.
 - 2) Otherwise, let *Y* be *YP*.
 - iii) Case:
 - 1) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.
 - 2) If the length in characters *LY* of *Y* is less than *LTD*, then *TV* is *Y* extended on the right by *LTD-LY* <space>s.
 - 3) Otherwise, an exception condition is raised: *data exception — string data, right truncation (22001)*.
 - b) If *SD* is an approximate numeric type, then:

- i) Let YP be a character string determined as follows

Case:

- 1) If SV equals 0 (zero), then YP is '0E0'.
- 2) Otherwise, YP is the shortest character string that conforms to the definition of <approximate numeric literal> in Subclause 20.1, "<literal>", whose interpreted value is equal to the absolute value of SV and whose <mantissa> consists of a single <digit> that is not '0' (zero), followed by a <period> and an <unsigned integer>.

- ii) Case:

- 1) If SV is less than 0 (zero), then let Y be the result of ' $-$ ' $\mid\mid$ YP .
- 2) Otherwise, let Y be YP .

- iii) Case:

- 1) If the length in characters LY of Y is equal to LTD , then TV is Y .
- 2) If the length in characters LY of Y is less than LTD , then TV is Y extended on the right by $LTD-LY$ <space>s.
- 3) Otherwise, an exception condition is raised: *data exception — string data, right truncation (22001)*.

- c) If SD is a fixed-length character string type or a variable-length character string type, then

Case:

- i) If the length in characters of SV is equal to LTD , then TV is SV .
- ii) If the length in characters of SV is larger than LTD , then TV is the first LTD characters of SV . If any of the remaining characters of SV are non-<whitespace> characters, then a completion condition is raised: *warning — string data, right truncation (01004)*.
- iii) If the length in characters M of SV is smaller than LTD , then TV is SV extended on the right by $LTD-M$ <space>s.

- d) If SD is a temporal instant type or a duration type, then:

- i) Let Y be the shortest character string that conforms to the definition of <literal> in Subclause 20.1, "<literal>", and such that the interpreted value of Y is SV , and such that

Case:

- 1) If SD is a date type, the character string includes the time scale components: [year], [month] and [day].
- 2) If SD is a localtime type, the character string includes the time scale components: [hour], [min] and [sec], but does not include the 'T' time designator.
- 3) If SD is a time type, the character string includes the time scale components: [hour], [min], [sec], and [shift], but does not include the 'T' time designator.
- 4) If SD is a localdatetime type, the character string includes the time scale components: [year], [month], [day], [hour], [min] and [sec].
- 5) If SD is a datetime type, the character string includes the time scale components: [year], [month], [day], [hour], [min], [sec], and [shift].

ii) Case:

- 1) If the length in characters LY of Y is equal to LTD , then TV is Y .
- 2) If the length in characters LY of Y is less than LTD , then TV is Y extended on the right by $LTD-LY$ <space>s.
- 3) Otherwise, an exception condition is raised: *data exception — string data, right truncation (22001)*.

e) If SD is a Boolean type, then

Case:

- i) If SV is *True* and LTD is not less than 4, then TV is 'TRUE' extended on the right by $LTD-4$ <space>s.
- ii) If SV is *False* and LTD is not less than 5, then TV is 'FALSE' extended on the right by $LTD-5$ <space>s.
- iii) Otherwise, an exception condition is raised: *data exception — invalid character value for cast (22018)*.

7) If TD is a variable-length character string type, then let $MLTD$ be the maximum length in characters of TD .

Case:

a) If SD is an exact numeric type, then:

- i) Let YP be the shortest character string that conforms to the definition of <exact numeric literal> in Subclause 20.1, “<literal>”, whose scale is the same as the scale of SD , whose interpreted value is the absolute value of SV , and that is not an <unsigned hexadecimal integer>.

ii) Case:

- 1) If SV is less than 0 (zero), then let Y be the result of ' $-$ ' || YP .
- 2) Otherwise, let Y be YP .

iii) Case:

- 1) If the length in characters LY of Y is less than or equal to $MLTD$, then TV is Y .
- 2) Otherwise, an exception condition is raised: *data exception — string data, right truncation (22001)*.

b) If SD is an approximate numeric type, then:

- i) Let YP be a character string determined as follows

Case:

- 1) If SV equals 0 (zero), then YP is '0E0'.

- 2) Otherwise, YP is the shortest character string that conforms to the definition of <approximate numeric literal> in Subclause 20.1, “<literal>”, whose interpreted value is equal to the absolute value of SV and whose <mantissa> consists of a single <digit> that is not '0', followed by a <period> and an <unsigned integer>.

ii) Case:

- 1) If SV is less than 0 (zero), then let Y be the result of ' $-$ ' || YP .

- 2) Otherwise, let Y be YP .
 - iii) Case:
 - 1) If the length in characters LY of Y is less than or equal to $MLTD$, then TV is Y .
 - 2) Otherwise, an exception condition is raised: *data exception — string data, right truncation (22001)*.
 - c) If SD is a fixed-length character string type, or a variable-length character string type, then

Case:

 - i) If the length in characters of SV is less than or equal to $MLTD$, then TV is SV .
 - ii) If the length in characters of SV is larger than $MLTD$, then TV is the first $MLTD$ characters of SV . If any of the remaining characters of SV are non-<whitespace> characters, then a completion condition is raised: *warning — string data, right truncation (01004)*.
 - d) If SD is a temporal instant type or a duration type, then:
 - i) Let Y be the shortest character string that conforms to the definition of <literal> in **Subclause 20.1, “<literal>”**, and such that the interpreted value of Y is SV , and such that

Case:

 - 1) If SD is a date type, the character string includes the time scale components: [year], [month] and [day].
 - 2) If SD is a localtime type, the character string includes the time scale components: [hour], [min] and [sec], but does not include the 'T' time designator.
 - 3) If SD is a time type, the character string includes the time scale components: [hour], [min], [sec], and [shift], but does not include the 'T' time designator.
 - 4) If SD is a localdatetime type, the character string includes the time scale components: [year], [month], [day], [hour], [min] and [sec].
 - 5) If SD is a datetime type, the character string includes the time scale components: [year], [month], [day], [hour], [min], [sec], and [shift].
 - ii) Case:
 - 1) If the length in characters LY of Y is less than or equal to $MLTD$, then TV is Y .
 - 2) Otherwise, an exception condition is raised: *data exception — string data, right truncation (22001)*.
 - e) If SD is a Boolean type, then

Case:

 - i) If SV is *True* and $MLTD$ is not less than 4, then TV is 'TRUE'.
 - ii) If SV is *False* and $MLTD$ is not less than 5, then TV is 'FALSE'.
 - iii) Otherwise, an exception condition is raised: *data exception — invalid character value for cast (22018)*.
- 8) If TD is the date type, then
- Case:

19.22 <cast specification>

- a) If SD is character string type, then SV is replaced by SV with any leading or trailing <whitespace> removed.

Case:

- i) If the rules for <literal> in Subclause 20.1, “<literal>”, can be applied to SV to determine a valid value of the value type TD , then let TV be that value.
- ii) Otherwise, an exception condition is raised: *data exception — invalid datetime format (22007)*.

- b) If SD is the date type, then TV is SV .

- c) If SD is the datetime type or localdatetime type, then let D be the result of $\text{SUBSTRING} (\text{CAST} (VE \text{ AS } \text{STRING}), 1, 8)$. TV is the result of $\text{DATE} (DS)$.

- 9) Let STZ be the time zone identifier of the current session context. If STZ is ‘Z’, then let $STZD$ be “ $\text{DURATION} ('P0000')$; otherwise let $STZD$ be “ $\text{DURATION} ('P' || \text{SUBSTRING} (STZ, 1, 3) || 'H' || \text{SUBSTRING} (STZ, 4, 2) || 'M')$ ”.

- 10) If TD is the localtime type, then

Case:

- a) If SD is a character string type, then SV is replaced by SV with any leading or trailing <whitespace> removed.

Case:

- i) If the rules for <literal> in Subclause 20.1, “<literal>”, can be applied to SV to determine a valid value of the value type TD , then let TV be that value.
- ii) Otherwise, an exception condition is raised: *data exception — invalid datetime format (22007)*.

- b) If SD is a localtime type, then TV is SV .

- c) If SD is a time type, then:

- i) Let $TIME$ be the result of $\text{CAST} (VE \text{ AS } \text{STRING})$. Let TL be the result of $\text{CHARACTER_LENGTH} (TIME)$. If $\text{SUBSTRING} (TIME, TL - 1) = 'Z'$, then $TIME$ is replaced by $\text{SUBSTRING} (TIME, 1, TL - 1) || '+0000'$. Let TZ be the result of $\text{SUBSTRING} (TIME, TL - 4)$. Let TZD be “ $\text{DURATION} ('P' || \text{SUBSTRING} (TZ, 1, 3) || 'H' || \text{SUBSTRING} (TZ, 4, 2) || 'M')$ ”. Let T be the result of $\text{SUBSTRING} (TIME, 1, TL - 5)$.

- ii) TV is the result of $\text{LOCALTIME} (T) + TZD$.

- d) If SD is localdatetime type, then:

- i) Let T be the result of $\text{SUBSTRING} (\text{CAST} (VE \text{ AS } \text{STRING}), 10)$.
- ii) TV is the result of $\text{LOCALTIME} (T)$.

- e) If SD is datetime type, then:

- i) Let $TIME$ be the result of $\text{SUBSTRING} (\text{CAST} (VE \text{ AS } \text{STRING}), 10)$. Let TL be the result of $\text{CHARACTER_LENGTH} (TIME)$. If $\text{SUBSTRING} (TIME, TL - 1) = 'Z'$, then $TIME$ is replaced by $\text{SUBSTRING} (TIME, 1, TL - 1) || '+0000'$. Let TZ be the result of $\text{SUBSTRING} (TIME, TL - 4)$. Let TZD be “ $\text{DURATION} ('P' || \text{SUBSTRING} (TZ, 1, 3) || 'H' || \text{SUBSTRING} (TZ, 4, 2) || 'M')$ ”. Let T be the result of $\text{SUBSTRING} (TIME, 1, TL - 5)$.

ii) TV is the result of `LOCALTIME (T) + TZD`.

11) If TD is the time type, then

Case:

- a) If SD is a character string type, then SV is replaced by SV with any leading or trailing `<whitespace>` removed.

Case:

i) If the rules for `<literal>` in Subclause 20.1, “`<literal>`”, can be applied to SV to determine a valid value of the value type TD , then let TV be that value.

ii) Otherwise, an exception condition is raised: *data exception — invalid datetime format (22007)*.

- b) If SD is a time type, then TV is SV .

- c) If SD is a localtime type, then:

i) Let T be the result of `CAST (VE AS STRING)`.

ii) TV is the result of `TIME (T || STZ)`.

- d) If SD is datetime type, then:

i) Let T be the result of `SUBSTRING (CAST (VE AS STRING), 10)`.

ii) TV is the result of `TIME (T)`.

- e) If SD is localdatetime type, then:

i) Let T be the result of `SUBSTRING (CAST (VE AS STRING), 10)`.

ii) TV is the result of `TIME (T || STZ)`.

12) If TD is the localdatetime type, then

Case:

- a) If SD is a character string type, then SV is replaced by SV with any leading or trailing `<whitespace>` removed.

Case:

i) If the rules for `<literal>` in Subclause 20.1, “`<literal>`”, can be applied to SV to determine a valid value of the value type TD , then let TV be that value.

ii) Otherwise, an exception condition is raised: *data exception — invalid datetime format (22007)*.

- b) If SD is a date type, then let D be the result of `CAST (VE AS STRING)`. TV is the result of `LOCALDATETIME (D || 'T000000')`.

- c) If SD is a localtime type, then:

i) Let CD be the result of `CAST (DATE() AS STRING)`.

ii) Let T be the result of `CAST (VE AS STRING)`.

iii) TV is the result of `LOCALDATETIME (CD || 'T' || T)`.

- d) If SD is a time type, then:

19.22 <cast specification>

- i) Let TZ be the result of `SUBSTRING (CAST (VE AS STRING), 6)`. If TZ is 'Z', then let TZD be "DURATION ('P0000')"; otherwise let TZD be "DURATION ('P' || SUBSTRING (TZ, 1, 3) || 'H' || SUBSTRING (TZ, 4, 2) || 'M')".
 - ii) Let T be the result of `SUBSTRING (CAST (VE AS STRING), 1, 6)`.
 - iii) Let CD be the result of `CAST (DATE() AS STRING)`.
 - iv) Let TA be the result of `CAST (LOCALTIME (T) + TZD AS STRING)`.
 - v) TV is the result of `LOCALDATETIME (CD || 'T' || TA)`.
 - e) If SD is a localdatetime type, then TV is SV .
 - f) If SD is datetime type, then:
 - i) Let TZ be the result of `SUBSTRING (CAST (VE AS STRING), 16)`. If TZ is 'Z', then let TZD be DURATION ('P0000'); otherwise let TZD be DURATION ('P' || SUBSTRING (TZ, 1, 3) || 'H' || SUBSTRING (TZ, 4, 2) || 'M').
 - ii) Let DT be the result of `SUBSTRING (CAST (VE AS STRING), 1, 15)`.
 - iii) TV is the result of `LOCALDATETIME (DT) + TZD`.
- 13) If TD is the datetime type, then
- Case:
- a) If SD is a character string type, then SV is replaced by SV with any leading or trailing <whitespace> removed.
- Case:
- i) If the rules for <literal> in Subclause 20.1, "<literal>", can be applied to SV to determine a valid value of the value type TD , then let TV be that value.
 - ii) Otherwise, an exception condition is raised: *data exception — invalid datetime format (22007)*.
 - b) If SD is a date type, then let D be the result of `CAST (VE AS STRING)`. TV is the result of `DATETIME (D || 'T000000' || STZ)`.
 - c) If SD is a localtime type, then:
 - i) Let CD be the result of `CAST (DATE() AS STRING)`.
 - ii) Let T be the result of `CAST (VE AS STRING)`.
 - iii) TV is the result of `DATETIME (CD || 'T' || T || STZ)`.
 - d) If SD is a time type, then:
 - i) Let CD be the result of `CAST (DATE() AS STRING)`.
 - ii) Let T be the result of `CAST (VE AS STRING)`.
 - iii) TV is the result of `DATETIME (CD || 'T' || T)`.
 - e) If SD is localdatetime type, then:
 - i) Let DT be the result of `CAST (VE AS STRING)`.
 - ii) TV is the result of `DATETIME (DT || STZ)`.
 - f) If SD is a datetime type, then TV is SV .

14) If TD is a duration type, then

Case:

- a) If SD is a character string type, then SV is replaced by SV with any leading or trailing $<\text{whitespace}>$ removed.

Case:

- i) If the rules for $<\text{literal}>$ in Subclause 20.1, “ $<\text{literal}>$ ”, can be applied to SV to determine a valid value of the value type TD , then let TV be that value.
- ii) Otherwise, an exception condition is raised: *data exception — invalid duration format (22G0H)*.

- b) If SD is a duration type, then TV is SV .

15) If TD is Boolean type, then

Case:

- a) If SD is a character string type, then SV is replaced by SV with any leading or trailing $<\text{whitespace}>$ removed.

Case:

- i) If the rules for $<\text{literal}>$ in Subclause 20.1, “ $<\text{literal}>$ ”, can be applied to SV to determine a valid value of the value type TD , then let TV be that value.
- ii) Otherwise, an exception condition is raised: *data exception — invalid character value for cast (22018)*.

- b) If SD is a Boolean type, then TV is SV .

16) If TD and SD are byte string types, then:

- a) If TD is a byte string type, then let MINLTD be the minimum length in bytes of TD and let MAXLTD be the maximum length in bytes of TD .

b) Case:

- i) If the length in bytes of SV is equal to LTD , then TV is SV .
- ii) If the length in bytes of SV is larger than MAXLTD , then TV is the first MAXLTD bytes of SV and a completion condition is raised: *warning — string data, right truncation (01004)*.
- iii) If the length in bytes M of SV is smaller than MINLTD , then TV is SV extended on the right by $\text{MINLTD} - M$ X'00's.

17) The result of CS is TV .

Conformance Rules

None.

19.23 <element_id function>

Function

Generate a unique identifier for a graph element.

Format

```
<element_id function> ::=  
ELEMENT_ID <left paren> <element reference> <right paren>
```

Syntax Rules

- 1) The <element reference> shall have singleton degree of reference.
- 2) The declared type of <element_id function> is an implementation-defined ([ID076](#)) type that is permitted as the declared type of an operand of an equality operation according to the Syntax Rules of [Subclause 21.8, “Equality operations”](#).

**** Editor's Note (number 252) ****

This rule differs from that in SQL/PGQ. The SQL/PGQ text adds “and as the declared type of an operand of a grouping operation according to the Syntax Rules of [Subclause 9.12, “Grouping operations”](#)”. Since GQL currently does not have any of the data types excluded by the above Subclause and only one active collation, there appears to be no need for the additional restriction. This should be kept under review.

General Rules

- 1) Let LOE be the list of graph elements bound to the <element reference>.
 - 2) Case:
 - a) If LOE is empty, then the result of <element_id function> is the null value.
« WG3:BER-040R3 »
 - b) Otherwise, let GE be the sole graph element in LOE . The result of <element_id function> is an implementation-dependent ([UV004](#)) value that encapsulates the identity of GE in the graph that contains GE for the duration of the currently executing GQL-request.
- NOTE 145 — By implication, the value returned as the result of <element_id function> may be but is not guaranteed to be the same as the global object identifier of GE .

Conformance Rules

- 1) Without Feature G100, “ELEMENT_ID function”, conforming GQL language shall not contain an <element_id function>.

20 Lexical elements

**** Editor's Note (number 253) ****

This clause is based on Clause 5, "Lexical elements" in SQL/Foundation and adapted by [WG3:MMX-011r1], [WG3:MMX-028r2] and [WG3:W03-034]. Two proposals, [WG3:W04-015] and [WG3:W04-019], attempted to resolve perceived issues with this clause and both were rejected by WG3. WG3:W05-020 suggests that further discussion of this clause is needed to establish consensus. See Possible Problem **GQL-086**.

20.1 <literal>

Function

Specify a value.

Format

```

<literal> ::= 
    <signed numeric literal>
  | <general literal>
  « WG3:BER-038R1 »
  « WG3:BER-094R1 »

<general literal> ::= 
    <predefined type literal>
  | <list literal>
  | <map literal>
  | <record literal>

<predefined type literal> ::= 
    <boolean literal>
  | <character string literal>
  | <byte string literal>
  | <temporal literal>
  | <duration literal>
  | <null literal>

<unsigned literal> ::= 
    <unsigned numeric literal>
  | <general literal>

<boolean literal> ::= 
    TRUE | FALSE | UNKNOWN

<character string literal> ::= 
    <single quoted character sequence>
  | <double quoted character sequence>

<unbroken character string literal> ::= 
    <unbroken single quoted character sequence>
  | <unbroken double quoted character sequence>

<single quoted character sequence> ::= 
    <unbroken single quoted character sequence>

```

20.1 <literal>

```

[ { <separator> <unbroken single quoted character sequence> }... ]

<double quoted character sequence> ::==
  <unbroken double quoted character sequence>
    [ { <separator> <unbroken double quoted character sequence> }... ]

<unbroken single quoted character sequence> ::==
  <quote> [ <single quoted character representation>... ] <quote>

<unbroken double quoted character sequence> ::==
  <double quote> [ <double quoted character representation>... ] <double quote>

<unbroken accent quoted character sequence> ::==
  <grave accent> [ <accent quoted character representation>... ] <grave accent>

<single quoted character representation> ::==
  <character representation>
  !! See the Syntax Rules.

<double quoted character representation> ::==
  <character representation>
  !! See the Syntax Rules.

<accent quoted character representation> ::==
  <character representation>
  !! See the Syntax Rules.

<character representation> ::==
  <string literal character>
  | <escaped character>

<string literal character> ::==
  !! See the Syntax Rules.

<escaped character> ::==
  <escaped reverse solidus>
  | <escaped quote>
  | <escaped double quote>
  | <escaped grave accent>
  | <escaped tab>
  | <escaped backspace>
  | <escaped newline>
  | <escaped carriage return>
  | <escaped form feed>
  | <unicode escape value>

<escaped reverse solidus> ::==
  <reverse solidus> <reverse solidus>

<escaped quote> ::==
  <reverse solidus> <quote>

<escaped double quote> ::==
  <reverse solidus> <double quote>

<escaped grave accent> ::==
  <reverse solidus> <grave accent>
  | <doubled grave accent>

<doubled grave accent> ::==
  `` !! <U+0060, U+0060>

<escaped tab> ::==
  <reverse solidus> t

```

```

<escaped backspace> ::= 
    <reverse solidus> b

<escaped newline> ::= 
    <reverse solidus> n

<escaped carriage return> ::= 
    <reverse solidus> r

<escaped form feed> ::= 
    <reverse solidus> f

<unicode escape value> ::= 
    <unicode 4 digit escape value>
    | <unicode 6 digit escape value>

<unicode 4 digit escape value> ::= 
    <reverse solidus> u <hex digit> <hex digit> <hex digit> <hex digit>

<unicode 6 digit escape value> ::= 
    <reverse solidus> U <hex digit> <hex digit> <hex digit> <hex digit> <hex digit>
    <hex digit>

<byte string literal> ::= 
    X <quote> [ <space>... ] [ { <hex digit> [ <space>... ] <hex digit> [ <space>... ] }...
        ] <quote>
        [ { <separator> <quote> [ <space>... ] [ { <hex digit> [ <space>... ]
            <hex digit> [ <space>... ] }... ] <quote> }... ]

<signed numeric literal> ::= 
    [ <sign> ] <unsigned numeric literal>

<unsigned numeric literal> ::= 
    <exact numeric literal>
    | <approximate numeric literal>

<exact numeric literal> ::= 
    <unsigned integer>
    | <unsigned decimal integer> [ <period> [ <unsigned decimal integer> ] ]
    | <period> <unsigned decimal integer>

<sign> ::= 
    <plus sign>
    | <minus sign>

<unsigned integer> ::= 
    <unsigned decimal integer>
    | <unsigned hexadecimal integer>
    | <unsigned octal integer>
    | <unsigned binary integer>

<unsigned decimal integer> ::= 
    <digit> [ { [ <underscore> ] <digit> }... ]

<unsigned hexadecimal integer> ::= 
    0x { [ <underscore> ] <hex digit> }...

<unsigned octal integer> ::= 
    0o { [ <underscore> ] <octal digit> }...

<unsigned binary integer> ::= 
    0b { [ <underscore> ] <binary digit> }...

<signed decimal integer> ::= 
    [ <sign> ] <unsigned decimal integer>

```

20.1 <literal>

```

<approximate numeric literal> ::= 
  <mantissa> E <exponent>

<mantissa> ::= 
  <exact numeric literal>

<exponent> ::= 
  <signed decimal integer>

<temporal literal> ::= 
  <date literal>
  | <time literal>
  | <datetime literal>

<date literal> ::= 
  DATE <date string>

<time literal> ::= 
  TIME <time string>

<datetime literal> ::= 
  { DATETIME | TIMESTAMP } <datetime string>

<date string> ::= 
  <unbroken character string literal>

<time string> ::= 
  <unbroken character string literal>

<datetime string> ::= 
  <unbroken character string literal>

<duration literal> ::= 
  DURATION <duration string>
  | <SQL-interval literal>

<duration string> ::= 
  <unbroken character string literal>

<SQL-interval literal> ::= 
  !! See the Syntax Rules.

<null literal> ::= 
  NULL

<list literal> ::= 
  <list value constructor by enumeration>
  « WG3:BER-094R1 deleted two productions »
  « WG3:BER-038R1 deleted one production »

<map literal> ::= 
  <map value constructor by enumeration>
  « WG3:BER-040R3 »

<record literal> ::= 
  <record value constructor>

```

Syntax Rules

- 1) An <unsigned decimal integer> that immediately contains <underscore>s is equivalent to the same <unsigned decimal integer> with every <underscore> removed.

- 2) An <unsigned hexadecimal integer> that immediately contains <underscore>s is equivalent to the same <unsigned hexadecimal integer> with every <underscore> removed.
- 3) An <unsigned octal integer> that immediately contains <underscore>s is equivalent to the same <unsigned octal integer> with every <underscore> removed.
- 4) An <unsigned binary integer> that immediately contains <underscore>s is equivalent to the same <unsigned binary integer> with every <underscore> removed.
- 5) An <unsigned hexadecimal integer> is equivalent to an <unsigned decimal integer> containing a series of <digit>s that represent the same integer value as the series of <hex digit>s.
- 6) An <unsigned octal integer> is equivalent to an <unsigned decimal integer> containing a series of <digit>s that represent the same integer value as the series of <octal digit>s.
- 7) An <unsigned binary integer> is equivalent to an <unsigned decimal integer> containing a series of <digit>s that represent the same integer value as the series of <binary digit>s.
- 8) A <mantissa> shall not contain an <unsigned integer> that is not an <unsigned decimal integer>.
- 9) The maximum number of <digit>s immediately contained in an <unsigned integer> is implementation-defined ([ID078](#)) but shall not be less than 9.
- 10) An <exact numeric literal> without a <period> has an implicit <period> following the last <digit>.
- 11) The declared type of an <exact numeric literal> *ENL* is an exact numeric type whose scale is the number of <digit>s to the right of the <period>. There shall be an exact numeric type capable of representing the value of *ENL* exactly.
- 12) The declared type of an <approximate numeric literal> *ANL* is an implementation-defined ([ID079](#)) approximate numeric type. The value of *ANL* shall not be greater than the maximum value nor less than the minimum value that can be represented by the approximate numeric type.
- 13) The declared type of a <boolean literal> is the Boolean type.
- 14) The <character string literal> specifies the character string specified by the <single quoted character sequence> or the <double quoted character sequence> that it contains.
- 15) The <unbroken character string literal> specifies the character string specified by the <unbroken single quoted character sequence> or the <unbroken double quoted character sequence> that it contains.
- 16) The <single quoted character sequence> specifies the character string comprising the quote-separated concatenation of the character strings specified by the <unbroken single quoted character sequence>s that it contains.
- 17) The <double quoted character sequence> specifies the character string comprising the double quote-separated concatenation of the character strings specified by the <unbroken double quoted character sequence>s that it contains.
- 18) The <unbroken single quoted character sequence> specifies the character string comprising the sequence of characters defined by the <single quoted character representation>s that it contains.
- 19) The <unbroken double quoted character sequence> specifies the character string comprising the sequence of characters defined by the <double quoted character representation>s that it contains.
- 20) The <unbroken accent quoted character sequence> specifies the character string comprising the sequence of characters defined by the <accent quoted character representation>s that it contains.
- 21) The <single quoted character representation> shall not be a <quote> or a <reverse solidus> unless either of these occurs as part of an <escaped character>.

- 22) The <double quoted character representation> shall not be a <double quote> or a <reverse solidus> unless either of these occurs as part of an <escaped character>.
- 23) The <accent quoted character representation> shall not be a <grave accent> or a <reverse solidus> unless either of these occurs as part of an <escaped character>.
- 24) A <string literal character> is any character except for those occurring as part of an <escaped character>.
- 25) In an <escaped character> each <escaped reverse solidus> represents a <reverse solidus> character.
- 26) In an <escaped character> each <escaped quote> represents a <quote> character.
- 27) In an <escaped character> each <escaped double quote> represents a <double quote> character.
- 28) In an <escaped character> each <escaped grave accent> represents a <grave accent> character.
- 29) In an <escaped character> each <escaped tab> represents the Unicode character identified by the code point U+0009.
- 30) In an <escaped character> each <escaped backspace> represents the Unicode character identified by the code point U+0008.

**** Editor's Note (number 254) ****

The editor wonders what the point of this is. See Possible Problem [GQL-190](#).

- 31) In an <escaped character> each <escaped newline> represents the Unicode character identified by the code point U+000A.
- 32) In an <escaped character> each <escaped carriage return> represents the Unicode character identified by the code point U+000D.
- 33) In an <escaped character> each <escaped form feed> represents the Unicode character identified by the code point U+000C.
- 34) In an <escaped character> each <unicode escape value> represents the Unicode character identified by the code point.
- 35) The declared type of a <character string literal> is character string.
- 36) In a <byte string literal>, the sequence

```
<quote> [ <space>... ] { <hex digit> [ <space>... ]
<hex digit> [ <space>... ] }... <quote>
```

is equivalent to the sequence

```
<quote> { <hex digit> <hex digit> }... <quote>
```

NOTE 146 — The <hex digit>s in the equivalent sequence are in the same sequence and relative sequence as in the original <byte string literal>.

- 37) In a <byte string literal>, the sequence

```
<quote> { <hex digit> <hex digit> }... <quote> <separator>
<quote> { <hex digit> <hex digit> }... <quote>
```

is equivalent to the sequence

```
<quote> { <hex digit> <hex digit> }... { <hex digit> <hex digit> }... <quote>
```

NOTE 147 — The <hex digit>s in the equivalent sequence are in the same sequence and relative sequence as in the original <byte string literal>.

- 38) In a <byte string literal>, the introductory 'X' may be represented either in upper-case (as 'X') or in lower-case (as 'x').
- 39) In a <character string literal>, or <byte string literal>, a <separator> shall contain a <newline>.
- 40) The declared type of a <byte string literal> is a byte string type. Each <hex digit> appearing in the literal is equivalent to a quartet of bits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F are interpreted as 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111, respectively. The <hex digit>s a, b, c, d, e, and f have respectively the same values as the <hex digit>s A, B, C, D, E, and F.
- 41) The character string specified by the <unbroken character string literal> immediately contained in <date string> shall conform to the representation specified in clause 5.2 "Date" of ISO 8601-1:2019 as extended by clauses 4.3 "Additional explicit forms" and 4.4 "Numerical extensions" of ISO 8601-2:2019.
- 42) The declared type of <date literal> is DATE.
- 43) The character string specified by the <unbroken character string literal> immediately contained in <time string> shall conform to the representation specified in clause 5.3 "Time of day" of ISO 8601-1:2019 as extended by clauses 4.3 "Additional explicit forms" and 4.4 "Numerical extensions" of ISO 8601-2:2019.
- 44) If the <time string> does not contain a representation of a time shift then the declared type of <time literal> is LOCALTIME, otherwise the declared type of <time literal> is TIME.
- 45) The character string specified by the <unbroken character string literal> immediately contained in <datetime string> shall conform to the representation specified in clause 5.4 "Date and time of day" of ISO 8601-1:2019 as extended by clauses 4.3 "Additional explicit forms" and 4.4 "Numerical extensions" of ISO 8601-2:2019 with the optional addition of a IANA Time Zone Database time zone designator enclosed between a <left bracket> and a <right bracket>. If both an IANA Time Zone Database time zone designator and a ISO 8601-1:2019 time shift specification are present then the IANA Time Zone Database time zone designator shall resolve to the same value as the ISO 8601-1:2019 time shift specification.

**** Editor's Note (number 255) ****

During the discussion of WG3:W08-014 (See WG3:W09-001) it was apparent that there was no consensus on the best way to specify a time zone. Should the standard allow both an explicit offset and an IANA time zone designator and should the combination also be allowed. This topic need to be revisited to establish consensus. See Possible Problem GQL-182.

- 46) If the <datetime string> does not contain a representation of a time shift then the declared type of <datetime literal> is LOCALDATETIME, otherwise the declared type of <datetime literal> is DATETIME.
- 47) The declared type of <duration literal> is DURATION.
- 48) The declared type DT of a <null literal> NS is defined as follows

Case:

**** Editor's Note (number 256) ****

The way in which the declared type of NULL is determined may require further adjustment (as per the discussion of RKE-048).

- a) If DT can be determined by the context in which NS appears, then NS is effectively replaced by CAST(NS AS DT).

NOTE 148 — In every such context, *NS* is uniquely associated with some expression or site of declared type *DT*, which thereby becomes the declared type of *NS*.

- b) Otherwise, it is implementation-defined (IA014) whether an exception condition is raised: *syntax error or access rule violation — invalid syntax (42001)*. If an exception condition is not raised, then *DT* is implementation-defined (ID085).
- 49) The <unbroken character string literal> immediately contained in <duration string> shall conform to the representation specified in clause 5.5.2 “Duration” of ISO 8601-1:2019 as extended by clauses 4.3 “Additional explicit forms” and 4.4 “Numerical extensions” of ISO 8601-2:2019.
- 50) <SQL-interval literal> shall conform to the Syntax Rules of <interval literal> in ISO/IEC 9075-2:202x.
- 51) If <SQL-interval literal> is specified, then:
- a) If <SQL-interval literal> contains a <years value> *y* then let *Y* be 'Y*y*', otherwise let *Y* be the zero-length character string.
 - b) If <SQL-interval literal> contains a <months value> *m* then let *M* be 'M*m*', otherwise let *M* be the zero-length character string.
 - c) If <SQL-interval literal> contains a <days value> *d* then let *D* be 'D*d*', otherwise let *D* be the zero-length character string.
 - d) If <SQL-interval literal> contains an <hours value> *h* then let *H* be 'H*h*', otherwise let *H* be the zero-length character string.
 - e) If <SQL-interval literal> contains a <minutes value> *mn* then let *MN* be 'M*mn*', otherwise let *MN* be the zero-length character string.
 - f) If <SQL-interval literal> contains a <seconds value> *s* then let *S* be 'S*s*', otherwise let *S* be the zero-length character string.
 - g) If <SQL-interval literal> contains an <hours value>, <minutes value>, or <seconds value> then let *T* be 'T', otherwise let *T* be the zero-length character string.
 - h) If <SQL-interval literal> contains at most one <minus sign>, then let *SN* be '-', otherwise let *SN* be the zero-length character string.

<SQL-interval literal> it is equivalent to the <duration literal>:

DURATION 'SNPYMDTHMNS'

- 52) Every <value expression> contained in a <list element> of the <list element list> of the <list value constructor by enumeration> contained in a <list literal> shall be a <literal>.
- 53) The declared type of <list literal> is the declared type of the immediately contained <list value constructor by enumeration>.
 « WG3:BER-094R1 deleted four SRs »
 « WG3:BER-038R1 deleted two SRs »
- 54) Every <value expression> contained in a <map element> of the <map element list> of the <map value constructor by enumeration> contained in a <map literal> shall be a <literal>.
- 55) The declared type of <map literal> is the declared type of the immediately contained <map value constructor by enumeration>.
- 56) Every <value expression> contained in a <field> of the <field list> of the <record value constructor> contained in a <record literal> shall be a <literal>.
 « WG3:BER-040R3 »

- 57) The declared type of <record literal> is the declared type of the immediately contained <record value constructor>.

General Rules

- 1) Except when it is contained in an <exact numeric literal>, the value of an <unsigned integer> is the numeric value determined by application of the normal mathematical interpretation of positional decimal notation to the series of <digit>s that constitutes the <unsigned integer>.
- 2) The value of an <exact numeric literal> is the numeric value determined by application of the normal mathematical interpretation of positional decimal notation to the source characters that constitute the <exact numeric literal>.
- 3) Let *ANL* be an <approximate numeric literal>. Let *ANDT* be the declared type of *ANL*. Let *ANV* be the product of the exact numeric value represented by the <mantissa> of *ANL* and the number obtained by raising the number 10 to the power of the exact numeric value represented by the <exponent> of *ANL*. If *ANV* is a value of *ANDT*, then the value of *ANL* is *ANV*; otherwise, the value of *ANL* is a value of *ANDT* obtained from *ANV* by rounding or truncation. The choice of whether to round or truncate is implementation-defined (IA005).
- 4) The <sign> in a <signed numeric literal> is a monadic arithmetic operator. The monadic arithmetic operators + and – specify monadic plus and monadic minus, respectively. If neither monadic plus nor monadic minus are specified in a <signed numeric literal>, or if monadic plus is specified, then the literal is positive. If monadic minus is specified in a <signed numeric literal>, then the literal is negative.
- 5) The truth value of a <boolean literal> is *True* if TRUE is specified, is *False* if FALSE is specified, and is *Unknown* if UNKNOWN is specified.
- 6) The value of a <character string literal> is the character string that it specifies.
- 7) The value of an <unbroken character string literal> is the character string that it specifies.
- 8) The value of a <byte string literal> is the byte string comprising sequence of bits defined by the <hex digit>s that it contains. Each <hex digit> appearing in the literal is equivalent to a quartet of bits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F are interpreted as 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111, respectively. The <hex digit>s a, b, c, d, e, and f have respectively the same values as the <hex digit>s A, B, C, D, E, and F.
- 9) If <date string> contains a representation with reduced precision then the lowest valid value for each of the omitted lower order time scale components is implicit.
- 10) The value of a <date literal> is a calendar date in the Gregorian calendar.
- 11) If <time string> contains a representation with reduced precision then value for each of the omitted lower order time scale components is 0 (zero).
- 12) The value of a <time literal> is a time of day. If the <time string> contained a representation of a time shift then the time shift information is preserved.
- 13) If <datetime string> contains a representation with reduced precision then value for each of the omitted lower order time scale components is 0 (zero).
- 14) The value of a <datetime literal> is a time. If the <datetime string> contained a representation of a time shift then the time shift information is preserved.
- 15) The value of a <duration literal> is a duration or a negative duration.

NOTE 149 — See clause 4.4.1.9 Duration of ISO 8601-2:2019 for the definition of a negative duration.

- 16) The value of a <null literal> is the null value.
- 17) The value of <list literal> is the value of the immediately contained <list value constructor by enumeration>.« WG3:BER-094R1 deleted two GRs »
« WG3:BER-038R1 deleted one GR »
- 18) The value of <map literal> is the value of the immediately contained <map value constructor by enumeration>.« WG3:BER-040R3 »
- 19) The value of <record literal> is the value of the immediately contained <record value constructor>.

Conformance Rules

None.

20.2 <value type>

Function

Specify a value type.

Format

** Editor's Note (number 257) **

Support for graph types and graph element types with and without attached schema information needs to be developed further.
See Possible Problem [GQL-215](#).

```
« WG3:BER-040R3 »
« WG3:BER-094R1 »

<value type> ::=

    ANY
  | <predefined type>
  | <reference value type>
  | <list value type>
  | <map value type>
  | <record type>
  | NOTHING

<of value type> ::=
  [ <of type> ] <value type>
« WG3:BER-040R3 »

<of type> ::=
  <double colon> | OF

<predefined type> ::=
  <boolean type>
  | <character string type>
  | <byte string type>
  | <numeric type>
  | <temporal type>

<boolean type> ::=
  BOOL | BOOLEAN

<character string type> ::=
  { STRING | VARCHAR } [ <left paren> <max length> <right paren> ]

<byte string type> ::=
  BYTES [ <left paren> [ <min length> <comma> ] <max length> <right paren> ]
  | BINARY [ <fixed length> ]
  | VARBINARY [ <max length> ]

<min length> ::=
  <unsigned decimal integer>

<max length> ::=
  <unsigned decimal integer>

<fixed length> ::=
  <unsigned decimal integer>

<numeric type> ::=
```

20.2 <value type>

```

<exact numeric type>
| <approximate numeric type>

<exact numeric type> ::= 
    <binary exact numeric type>
| <decimal exact numeric type>
« WG3:BER-067R1 »

<binary exact numeric type> ::= 
    <signed binary exact numeric type>
| <unsigned binary exact numeric type>

<signed binary exact numeric type> ::= 
    INT8
| INT16
| INT32
| INT64
| INT128
| INT256
| SMALLINT
| INT [ <left paren> <precision> <right paren> ]
| BIGINT
| [ SIGNED ] <verbose binary exact numeric type>
« WG3:BER-067R1 »

<unsigned binary exact numeric type> ::= 
    UINT8
| UINT16
| UINT32
| UINT64
| UINT128
| UINT256
| USMALLINT
| UINT [ <left paren> <precision> <right paren> ]
| UBIGINT
| UNSIGNED <verbose binary exact numeric type>

<verbose binary exact numeric type> ::= 
    INTEGER8
| INTEGER16
| INTEGER32
| INTEGER64
| INTEGER128
| INTEGER256
| SMALL INTEGER
| INTEGER [ <left paren> <precision> <right paren> ]
| BIG INTEGER

<decimal exact numeric type> ::= 
{ DECIMAL | DEC } <left paren> <precision> [ <comma> <scale> ] <right paren>

<precision> ::= 
<unsigned decimal integer>

<scale> ::= 
<unsigned decimal integer>

<approximate numeric type> ::= 
    FLOAT16
| FLOAT32
| FLOAT64
| FLOAT128
| FLOAT256

```

```
| FLOAT [ <left paren> <precision> [ <comma> <scale> ] <right paren> ]
| REAL
| DOUBLE [ PRECISION ]

<temporal type> ::==
    DATETIME
    | LOCALDATETIME
    | DATE
    | TIME
    | LOCALTIME
    | DURATION

« WG3:BER-040R3 »

<reference value type> ::==
    <node reference value type>
    | <edge reference value type>

<node reference value type> ::==
    <refined node reference value type>
    | <open node reference value type>

<refined node reference value type> ::==
    <node type definition>

<open node reference value type> ::==
    <node synonym>

<edge reference value type> ::==
    <refined edge reference value type>
    | <open edge reference value type>

<refined edge reference value type> ::==
    <edge type definition>

<open edge reference value type> ::==
    <edge synonym>
« WG3:BER-094R1 deleted one production »

<list value type> ::==
    <value type> <list value type name>

<list value type name> ::==
    LIST | ARRAY
« WG3:BER-094R1 deleted two productions »
« WG3:BER-038R1 deleted one production »

<map value type> ::==
    MAP <left angle bracket> <map key type> <comma> <value type> <right angle bracket>

<map key type> ::==
    <predefined type>
« WG3:BER-040R3 »

<record type> ::==
    [ RECORD ] <field type specification>

<field type specification> ::==
    <left brace> [ <field type list> ] <right brace>

<field type list> ::==
    <field type> [ { <comma> <field type> }... ]
« WG3:BER-040R3 deleted one production »
```

**** Editor's Note (number 258) ****

The collection data types needed by GQL need to be fully defined. See Possible Problem [GQL-006](#).

Syntax Rules

« WG3:BER-040R3 »

- 1) Every <value type> specifies the *value type* that is the data type specified by the immediately contained BNF element.
- 2) Every <predefined type> specifies the data type specified by the immediately contained BNF element.
- 3) Every <reference value type> specifies the *reference value type* that is the data type specified by the immediately contained <node reference value type> or <edge reference value type>.
- 4) The base type of Boolean types is named BOOLEAN DATA. The preferred name of Boolean types is implementation-defined ([ID021](#)) as either BOOLEAN or BOOL.
- 5) Every <boolean type> specifies a *Boolean type*.
- 6) For each Boolean type *BT*, there is a Boolean type *BNF(BT)*, known as the *normal form* of *BT* (which may be *BT* itself), such that the declared name of *BNF(BT)* is the preferred name of Boolean types.
- 7) The value of every <max length> shall be greater than or equal to 1 (one).
- 8) The base type of character string types is named STRING DATA. The preferred name of character string types is implementation-defined ([ID023](#)) as either VARCHAR or STRING.
- 9) Every <character string type> *CST* specifies a *character string type*.
- 10) If the <max length> *CSMAXL* is specified in a <character string type> *CST*, then the maximum length of the character string type specified by *CST* is the value of *CSMAXL*; otherwise, the maximum length of the character string type specified by *CST* is implementation-defined ([IL013](#)).
- 11) For each character string type *CST*, there is a character string type *CSNF(CST)*, known as the *normal form* of *CST* (which may be *CST* itself), such that *CSNF(CST)* and *CST* have the same maximum length and the declared name of *CSNF(CST)* is the preferred name of character string types.
- 12) Every character string type with maximum length *CSMAXL* only includes character strings with a length that is less than or equal to *CSMAXL*.
- 13) The maximum length of a character string is implementation-defined ([IL013](#)) but shall be greater than or equal to $2^{14}-1 = 16383$ characters.
- 14) Characters in a character string are numbered beginning with 1 (one).

**** Editor's Note (number 259) ****

Use of 1-based numbering vs. 0-based numbering to be decided. See Possible Problem [GQL-216](#).

- 15) The value of every <min length> shall be greater than or equal to 0 (zero).
- 16) If <min length> is omitted, then a <min length> of 0 (zero) is implicit.
- 17) The value of every <fixed length> shall be greater than or equal to 1 (one).
- 18) If <fixed length> is omitted, then a <fixed length> of 1 (one) is implicit.

- 19) The base type of byte string types is named BINARY DATA. The preferred name of fixed-length byte string types is implementation-defined (ID024) as either BINARY or BYTES. The preferred name of variable-length byte string types is implementation-defined (ID027) as either VARBINARY or BYTES.
- 20) Every <byte string type> specifies a *byte string type*.
- 21) If both the <min length> *MINL* and the <max length> *MAXL* are specified in a <byte string type>, then *MINL* shall be less than or equal to *MAXL*.
- 22) If the <min length> *BSMINL* is specified in a <byte string type> *BST*, then the minimum length of the byte string type specified by *BST* is the value of *BSMINL*.
- 23) If the <max length> *BSMAXL* is specified in a <byte string type> *BST*, then the maximum length of the byte string type specified by *BST* is the value of *BSMAXL*.
- 24) If the <fixed length> *BSFIXL* is specified in a <byte string type> *BST*, then the minimum length of the byte string type specified by *BST* is the value of *BSFIXL* and the maximum length of the byte string type specified by *BST* is the value of *BSFIXL*.
- 25) If neither the <max length> nor the <fixed length> are specified in a <byte string type> *BST*, then the maximum length of the byte string type specified by *BST* is implementation-defined (IL014).
- 26) For each byte string type *BST*, there is a byte string type *BSNF(BST)*, known as the *normal form* of *BST* (which may be *BST* itself), such that *BSNF(BST)* and *BST* have the same minimum length and the same maximum length and

Case:

- a) If *BST* is a fixed-length byte string type, then the declared name of *BSNF(BST)* is the preferred name of fixed-length byte string types.
- b) If *BST* is a variable-length byte string type, then the declared name of *BSNF(BST)* is the preferred name of variable-length byte string types.
- 27) Every byte string type with minimum length *BSMINL* only includes byte strings with a length that is greater than or equal to *BSMINL*.
- 28) Every byte string type with maximum length *BSMAXL* only includes byte strings with a length that is less than or equal to *BSMAXL*.
- 29) The minimum length of every byte string type *BST* shall be less than or equal to the maximum length of *BST*.
- 30) The maximum length of a byte string is implementation-defined (IL014) but shall be greater than or equal to $2^{16}-2=65534$ bytes.
- 31) Bytes in a byte string are numbered beginning with 1 (one).

** Editor's Note (number 260) **

Use of 1-based numbering vs. 0-based numbering to be decided. See Possible Problem [GQL-216](#).

- 32) The value of every <precision> shall be greater than or equal to 1 (one).
- 33) The base type of exact numeric types with binary precision is named INTEGER DATA. The base type of exact numeric types with decimal precision is named DECIMAL DATA.
- 34) The base type of approximate numeric types is named FLOAT DATA.

- 35) Every <numeric type> specifies the *numeric type* specified by the immediately contained <exact numeric type> or the immediately contained <approximate numeric type>.
- 36) For each exact numeric type *ENT*, there is an implementation-defined exact numeric type, *ENNF(ENT)*, known as the *normal form* of *ENT* (which may be *ENT* itself), such that:
- a) If *ENT1* and *ENT2* are exact numeric types whose declared names individually are either SIGNED INTEGER, INTEGER, or INT, then *ENNF(ENT1)* is the same as *ENNF(ENT2)*.

« WG3:BER-067R1 »

- b) If *ENT1* and *ENT2* are exact numeric types whose declared names individually are either SIGNED SMALL INTEGER, SMALL INTEGER, or SMALLINT, then *ENNF(ENT1)* is the same as *ENNF(ENT2)*.
- c) If *ENT1* and *ENT2* are exact numeric types whose declared names individually are either SIGNED BIG INTEGER, BIG INTEGER, or BIGINT, then *ENNF(ENT1)* is the same as *ENNF(ENT2)*.
- d) For each *p* from the set {16, 32, 64, 128, 256}: If *ENT1* and *ENT2* are exact numeric types whose declared names individually are either SIGNED INTEGER*p*, INTEGER*p*, or INT*p*, then *ENNF(ENT1)* is the same as *ENNF(ENT2)*.
- e) If *ENT1* and *ENT2* are exact numeric types whose declared names individually are either UNSIGNED INTEGER or UINT, then *ENNF(ENT1)* is the same as *ENNF(ENT2)*.

« WG3:BER-067R1 »

- f) If *ENT1* and *ENT2* are exact numeric types whose declared names individually are either UNSIGNED SMALL INTEGER, or USMALLINT, then *ENNF(ENT1)* is the same as *ENNF(ENT2)*.
- g) If *ENT1* and *ENT2* are exact numeric types whose declared names individually are either UNSIGNED BIG INTEGER, or UBIGINT, then *ENNF(ENT1)* is the same as *ENNF(ENT2)*.
- h) For each *p* from the set {16, 32, 64, 128, 256}: If *ENT1* and *ENT2* are exact numeric types whose declared names individually are either UNSIGNED INTEGER*p* or UINT*p*, then *ENNF(ENT1)* is the same as *ENNF(ENT2)*.
- i) If *ENT1* and *ENT2* are exact numeric types whose declared names individually are either DEC or DECIMAL, then *ENNF(ENT1)* is the same as *ENNF(ENT2)*.
- j) The precision, scale, and radix of *ENNF(ENT)* are the same as the precision, scale, and radix, respectively, of *ENT*.

NOTE 150 — The precision, scale, and radix are determined when an exact numeric type is specified prior to the construction of its descriptor.

- k) *ENNF(ENNF(ENT))* is the same as *ENNF(ENT)*.

- 37) For the <exact numeric type>s:

- a) The maximum value of a <precision> is implementation-defined (IL016). <precision> shall not be greater than this value.
- b) The maximum value of a <scale> is implementation-defined (IL017). <scale> shall not be greater than this maximum value.

- 38) Every <exact numeric type> specifies the *exact numeric type* specified by the immediately contained <binary exact numeric type> or the immediately contained <decimal exact numeric type>.

« WG3:BER-067R1 »

- 39) Every <binary exact numeric type> specifies the *binary exact numeric type* specified by the immediately contained <signed binary exact numeric type> or the immediately contained <unsigned binary exact numeric type>.

40) Every <signed binary exact numeric type> *BESNT* specifies a *signed binary exact numeric type*.

Case:

- a) If *BESNT* is SIGNED INTEGER8, INTEGER8, or INT8 then *BESNT* specifies the *signed 8-bit integer type* with precision of 7 and with scale 0 (zero).
- b) If *BESNT* is SIGNED INTEGER16, INTEGER16, or INT16 then *BESNT* specifies the *signed 16-bit integer type* with precision of 15 and with scale 0 (zero).
- c) If *BESNT* is SIGNED INTEGER32, INTEGER32, or INT32 then *BESNT* specifies the *signed 32-bit integer type* with precision of 31 and with scale 0 (zero).
- d) If *BESNT* is SIGNED INTEGER64, INTEGER64, or INT64 then *BESNT* specifies the *signed 64-bit integer type* with precision of 63 and with scale 0 (zero).
- e) If *BESNT* is SIGNED INTEGER128, INTEGER128, or INT128 then *BESNT* specifies the *signed 128-bit integer type* with precision of 127 and with scale 0 (zero).
- f) If *BESNT* is SIGNED INTEGER256, INTEGER256, or INT256 then *BESNT* specifies the *signed 256-bit integer type* with precision of 255 and with scale 0 (zero).
- g) If *BESNT* is SIGNED INTEGER, INTEGER, or INT, then *BESNT* specifies the *signed regular integer type* with implementation-defined ([ID028](#)) precision greater than or equal to 31 and with scale 0 (zero).
- h) If *BESNT* is SIGNED SMALL INTEGER, SMALL INTEGER, or SMALLINT, then *BESNT* specifies the *signed small integer type* with implementation-defined ([ID029](#)) precision less than or equal to the precision of the signed regular integer type and with scale 0 (zero).
- i) If *BESNT* is SIGNED BIG INTEGER, BIG INTEGER, or BIGINT, then *BESNT* specifies the *signed big integer type* with implementation-defined ([ID030](#)) precision greater than or equal to the precision of the signed regular integer type and with scale 0 (zero).
- j) Otherwise, *BESNT* specifies the *signed user-specified integer type* with implementation-defined ([ID031](#)) precision greater than or equal to the value of the <precision> that is immediately contained in *BESNT* as its binary precision in bits and with scale 0 (zero).

« WG3:BER-067R1 »

41) Every <unsigned binary exact numeric type> *BEUNT* specifies an *unsigned binary exact numeric type*.

Case:

- a) If *BEUNT* is UNSIGNED INTEGER8 or UINT8, then *BEUNT* specifies the *unsigned 8-bit integer type* with precision of 7 and with scale 0 (zero).
- b) If *BEUNT* is UNSIGNED INTEGER16 or UINT16, then *BEUNT* specifies the *unsigned 16-bit integer type* with precision of 15 and with scale 0 (zero).
- c) If *BEUNT* is UNSIGNED INTEGER32 or UINT32, then *BEUNT* specifies the *unsigned 32-bit integer type* with precision of 31 and with scale 0 (zero).
- d) If *BEUNT* is UNSIGNED INTEGER64 or UINT64, then *BEUNT* specifies the *unsigned 64-bit integer type* with precision of 63 and with scale 0 (zero).
- e) If *BEUNT* is UNSIGNED INTEGER128 or UINT128, then *BEUNT* specifies the *unsigned 128-bit integer type* with precision of 127 and with scale 0 (zero).
- f) If *BEUNT* is UNSIGNED INTEGER256 or UINT256, then *BEUNT* specifies the *unsigned 256-bit integer type* with precision of 255 and with scale 0 (zero).

- g) If *BEUNT* is UNSIGNED INTEGER or UINT, then *BEUNT* specifies the *unsigned regular integer type* with the same precision as the regular signed integer type SIGNED INTEGER and with scale 0 (zero).
 - h) If *BESNT* is UNSIGNED SMALL INTEGER or USMALLINT, then *BESNT* specifies the *unsigned small integer type* with the same precision as the signed small integer type SIGNED SMALL INTEGER and with scale zero (0).
 - i) If *BESNT* is UNSIGNED BIG INTEGER or UBIGINT, then *BESNT* specifies the *unsigned big integer type* with the same precision as the signed big integer type SIGNED BIG INTEGER and with scale zero (0).
 - j) Otherwise, *BEUNT* specifies the *unsigned user-specified integer type* with implementation-defined ([ID033](#)) precision greater than or equal to the value of the <precision> that is immediately contained in *BEUNT* as its binary precision in bits and with scale 0 (zero).
- 42) The value of every <scale> shall be greater than or equal to 0 (zero).
- 43) If an <exact numeric type> *ENT* contains the <scale> *SCALE*, then the value of *SCALE* shall not be greater than the value of the <precision> contained in *ENT*.
- 44) Every <decimal exact numeric type> *DENT* specifies an exact numeric type with decimal precision.
- Case:
- a) If *DENT* immediately contains DECIMAL or DEC and the <precision> *PREC* and the <scale> *SCALE*, then *DENT* specifies the *user-specified decimal exact numeric type* with implementation-defined ([ID036](#)) decimal precision in digits greater than or equal to the decimal precision in digits specified by *PREC* and with implementation-defined ([ID080](#)) decimal scale in digits specified by *SCALE*.
 - b) If *DENT* immediately contains DECIMAL or DEC and the <precision> *PREC* but no <scale>, then *DENT* specifies the *user-specified decimal exact numeric type* with implementation-defined ([ID035](#)) decimal precision in digits greater than or equal to the decimal precision in digits specified by *PREC* and with scale 0 (zero).
 - c) Otherwise, *DENT* specifies the *regular decimal exact numeric type* with implementation-defined ([ID034](#)) decimal precision and with scale 0 (zero).
- 45) The value of every <precision> contained in an <approximate numeric type> shall be equal to or greater than 2.
- NOTE 151 — This accounts for the possible inclusion of a leading bit in the precision describing the size of the mantissa of an approximate numeric value that is implied by but not included in the underlying physical representation.
- 46) For each approximate numeric type *ANT*, there is an implementation-defined approximate numeric type *NNNF(ANT)*, known as the *normal form* of *ANT* (which may be *ANT* itself), such that:
- a) The precision and scale of *NNNF(ANT)* are the same as the precision and scale, respectively, of *ANT*.
- NOTE 152 — The precision and scale are determined when an exact numeric type is specified prior to the construction of its descriptor.
- b) If *ANT1* and *ANT2* are exact numeric types whose declared names individually are either DOUBLE or DOUBLE PRECISION, then *NNNF(ANT1)* is the same as *NNNF(ANT2)*.
 - c) *NNNF(NNNF(ANT))* is the same as *NNNF(ANT)*.
- 47) For the <approximate numeric type>s:

- a) The maximum value of a <precision> is implementation-defined (ID082). <precision> shall not be greater than this value.
- b) The maximum value of a <scale> is implementation-defined (ID083). <scale> shall not be greater than this maximum value.

48) Every <approximate numeric type> *ANT* specifies an *approximate numeric type*:

Case:

« WG3:BER-085 »

- a) If *ANT* is FLOAT16, then *ANT* specifies the *16-bit approximate numeric type* with precision 10 and with scale 5.
- b) If *ANT* is FLOAT32, then *ANT* specifies the *32-bit approximate numeric type* with precision of 23 and with scale 8.
- c) If *ANT* is FLOAT64, then *ANT* specifies the *64-bit approximate numeric type* with precision of 52 and with scale 11.
- d) If *ANT* is FLOAT128, then *ANT* specifies the *128-bit approximate numeric type* with precision of 112 and with scale 15.
- e) If *ANT* is FLOAT256, then *ANT* specifies the *256-bit approximate numeric type* with precision of 236 and with scale 19.
- f) If *ANT* is FLOAT, then *ANT* specifies the *regular approximate numeric type* with implementation-defined (ID037) precision greater than or equal to 23 and with implementation-defined (ID038) scale greater than or equal to 8.
- g) If *ANT* is REAL, then *ANT* specifies the *real approximate numeric type* with an implementation-defined (ID039) precision less than or equal to the precision of the regular approximate numeric type and with implementation-defined (ID040) scale.
- h) If *ANT* is DOUBLE or DOUBLE PRECISION, then *ANT* specifies the *double approximate numeric type* with implementation-defined (ID041) precision greater than or equal to the precision of the regular approximate numeric type and with implementation-defined (ID042) scale.
- i) Otherwise, *ANT* specifies a user-specified approximate numeric type:

Case:

- i) If *ANT* immediately contains the <precision> *PREC* but no scale, then *ANT* specifies the *user-specified approximate numeric type* with implementation-defined (ID043) precision greater than or equal to *PREC* and with implementation-defined (ID044) scale.
- ii) Otherwise, *ANT* immediately contains the <precision> *PREC* and the <scale> *SCALE*, then *ANT* specifies the user-specified approximate numeric type with implementation-defined (ID045) precision greater than or equal to *PREC* and with implementation-defined (ID046) scale greater than or equal to *SCALE*.

- 49) The preferred name of every numeric type is the declared name of its normal form.
- 50) If a <numeric type> *NT* contains the <precision> *PREC*, then *PREC* is the explicitly specified precision of the numeric type specified by *NT*.
- 51) If a <numeric type> *NT* contains the <scale> *SCALE*, then *SCALE* is the explicitly specified scale of the numeric type specified by *NT*.

** Editor's Note (number 261) **

The following Rules require further discussion. See Possible Problem [GQL-098](#).

- 52) DATETIME specifies the datetime type.
- 53) LOCALDATETIME specifies the localdatetime type.
- 54) DATE specifies the date type.
- 55) TIME specifies the time type.
- 56) LOCALTIME specifies the localtime type.
- 57) DURATION specifies the duration type.
 « [WG3:BER-040R3](#) »
- 58) The base type of node reference value types is named NODE REFERENCE. The preferred name of node reference value types is implementation-defined ([ID089](#)) as either NODE or VERTEX.
- 59) Every <node reference value type> specifies the node reference value type specified by its immediately contained <refined node reference value type> or <open node reference value type>.
- 60) Every <refined node reference value type> specifies the closed node reference value type whose refined object type is the node type identified by its immediately contained <node type definition>.
- 61) Every <open node reference value type> specifies the open node reference value type.
- 62) The base type of edge reference value types is named EDGE REFERENCE. The preferred name of edge reference value types is implementation-defined ([ID090](#)) as either EDGE or RELATIONSHIP.
- 63) Every <edge reference value type> specifies the edge reference value type specified by its immediately contained <refined edge reference value type> or <open edge reference value type>.
- 64) Every <refined edge reference value type> specifies the refined edge reference value type whose refined object type is the edge type identified by its immediately contained <edge type definition>.
- 65) Every <open edge reference value type> specifies the open edge reference value type.
- 66) PATH specifies the value type path.
- 67) LIST or ARRAY specifies a list value type with the element type specified by the <value type>.
 « [WG3:BER-094R1 deleted two SRs](#) »
 « [WG3:BER-038R1 deleted one SR](#) »
- 68) MAP specifies a map value type with the map key type specified by the <map key type> and the map value type specified by the <value type>.
 « [WG3:BER-040R3](#) »
- 69) The base type of record types is named RECORD DATA.
- 70) Every <record type> *RT* specifies the *record type* determined as follows.

Case:

 - a) If *RT* simply contains a <field type specification> *FTS*, then *RT* specifies a *closed record type* whose field types are specified by *FTS*.
 - b) Otherwise, *RT* specifies the *open record type*.
- 71) For each record type *RT*, there is an implementation-dependent ([UA011](#)) record type, *RNF(RT)*, known as the *normal form* of *RT* (which may be *RT* itself), such that:

- a) If $RT1$ and $RT2$ are two record types with the same characteristics, then $RNF(RT1) = RNF(RT2)$.
 - b) $RNF(RNF(RT)) = RNF(RT)$.
- 72) Every <field type specification> specifies the field types specified by the <field type list> that it immediately contains.
- 73) Every <field type list> specifies the field types that are specified by the <field type>s that it immediately contains.
- 74) The maximum number of <field type>s immediately contained in a <field type list> is the implementation-defined (IL019) maximum number of record fields.
« WG3:BER-040R3 deleted one SR »

General Rules

« WG3:BER-040R3 »

- 1) If <boolean type> is specified, then a Boolean data type descriptor is created for the specified Boolean type BT that describes BT and comprises:
 - a) The name of the base type of all Boolean types (BOOLEAN DATA).
 - b) The preferred name of BT .
 - c) An indication that BT includes the null value.**« WG3:BER-040R3 »**
- 2) If <character string type> is specified, then a character string data type descriptor is created for the specified character string type CST that describes CST and comprises:
 - a) The name of the base type of all character string types (STRING DATA).
 - b) The preferred name of CST .
 - c) An indication that CST includes the null value.
 - d) The maximum length in characters of CST .**« WG3:BER-040R3 »**
- 3) If <byte string type> is specified, then a byte string data type descriptor is created for the specified byte string type BST that describes BST and comprises:
 - a) The name of the base type of all byte string types (BINARY DATA).
 - b) The preferred name of BST .
 - c) An indication that BST includes the null value.
 - d) The minimum length in bytes of BST .
 - e) The maximum length in bytes of BST .**« WG3:BER-040R3 »**
- 4) If <exact numeric type> is specified, then a numeric data type descriptor is created for the specified exact numeric type ENT that describes ENT and comprises:
 - a) The name of the base type of ENT (INTEGER DATA or DECIMAL DATA).
 - b) The preferred name of ENT , which is the declared name of the normal form of ENT .

20.2 <value type>

- c) An indication that *ENT* includes the null value.
- d) The (implemented) precision of *ENT*.
- e) If *ENT* specifies an exact numeric type with decimal precision, then the (implemented) scale of *ENT*.
- f) An indication of whether the precision and the scale of *ENT* are expressed in binary or decimal terms.
- g) The explicit declared precision of *ENT*, if specified.
- h) The explicit declared scale of *ENT*, if specified.

« WG3:BER-040R3 »

- 5) If <approximate numeric type> is specified, then a numeric data type descriptor is created for the specified approximate numeric type *ANT* that describes *ANT* and comprises:
 - a) The name of the base type of *ANT* (FLOAT DATA).
 - b) The preferred name of *ANT*, which is the declared name of the normal form of *ANT*.
 - c) An indication that *ANT* includes the null value.
 - d) The (implemented) precision of *ANT*.
 - e) The (implemented) scale of *ANT*.
 - f) An indication that the precision and the scale are expressed in binary terms.
 - g) The explicit declared precision of *ANT*, if specified.
 - h) The explicit declared scale of *ANT*, if specified.
- 6) If <open node reference value type> is specified, then a node reference value data type descriptor is created for the specified open node reference value type *ONRVT* that describes *ONRVT* and comprises:
 - a) The reference base type name that is the implementation-defined ([ID091](#)) base type name of node reference value types (NODE REFERENCE or VERTEX REFERENCE)
 - b) The object base type name that is the implementation-defined ([ID092](#)) base type name of node types (NODE DATA or VERTEX DATA).
 - c) No refined object type.
 - d) An indication that *ONRVT* includes the null value.
- 7) If <refined node reference value type> is specified, then a node reference value data type descriptor is created for the specified closed node reference value type *CNRVT* that describes *CNRVT* and comprises:
 - a) The reference base type name that is the implementation-defined ([ID091](#)) base type name of node reference value types (NODE REFERENCE or VERTEX REFERENCE)
 - b) The object base type name that is the implementation-defined ([ID092](#)) base type name of node types (NODE DATA or VERTEX DATA).
 - c) The specified refined object type.
 - d) An indication that *CNRVT* includes the null value.

- 8) If <edge reference value type> is specified, then an edge reference value data type descriptor is created for the specified edge reference value type *ERVT* that describes *ERVT* and comprises:
 - a) The reference edge type name that is the implementation-defined (ID093) name of the edge reference value types (EDGE REFERENCE or RELATIONSHIP REFERENCE).
 - b) The base edge type as the object type of the edge reference value type.
 - c) An indication that *RT* includes the null value.
- 9) If <record type> is specified, then let *RT* be the specified record type.

Case:

- a) If *RT* is a closed record type, then a record data type descriptor is created for *RT* that describes *RT* and comprises:
 - i) The base type name of all record types (RECORD DATA).
 - ii) An indication that *RT* does not include records with additional fields.
 - iii) A field type descriptor for every <field type> simply contained in the <record type>, according to the Syntax Rules and General Rules of Subclause 20.4, “<field type>”, applied to the <field type>s in the order in which they were specified.
 - iv) An indication that *RT* includes the null value.
- b) If *RT* is the open record type, then a record data type descriptor is created for *RT* that describes *RT* and comprises:
 - i) The base type name of all record types (RECORD DATA).
 - ii) An indication that *RT* includes records with additional fields.
 - iii) An indication that *RT* includes the null value.

Conformance Rules

None.

« WG3:BER-040R3 »

20.3 <property value type>

Function

Define a property value type.

Format

```
<property value type> ::=  
  <value type>
```

Syntax Rules

- 1) Let PV be the <property value type>.
- 2) Let VT be the <value type> immediately contained in PV .
- 3) PV specifies the property value type that is specified by VT .
- 4) The property value type specified by PV shall be a supported property value type.

General Rules

None.

Conformance Rules

None.

20.4 <field type>

Function

Specify a field type.

Format

```
<field type> ::=  
  <field name> [ <of type> ] <value type>
```

Syntax Rules

- 1) Let *FTL* be the <field type list> that simply contains a <field type> *FT*.
- 2) The <field name> shall not be equivalent to the <field name> of any other <field type> simply contained in *FTL*.
- 3) The <field name> specifies the field name of *FT*.
- 4) The <value type> specifies the field value type of *FT*.

General Rules

- 1) A data type descriptor is created that describes the field value type of the field type being defined.
- 2) A field type descriptor is created that describes the field type being defined. The field type descriptor includes the following:
 - a) The field name.
 - b) The data type descriptor of the field value type.

Conformance Rules

- 1) Without Feature GD01, “Nested record types”, conforming language shall not contain a <field type> that simply contains a <record type>.

20.5 Names and identifiers

Function

Specify names.

Format

```

<object name> ::= 
  <identifier>

<schema name> ::= 
  <identifier>

<graph name> ::= 
  <identifier>
  « Consequence of WG3:BER-040R3 »

<element type name> ::= 
  <identifier>

<graph type name> ::= 
  <identifier>
  « WG3:BER-040R3 deleted one production »

<binding table name> ::= 
  <identifier>

<value name> ::= 
  <identifier>

<procedure name> ::= 
  <identifier>

<function name> ::= 
  <identifier>

<label name> ::= 
  <identifier>

<property name> ::= 
  <identifier>

<field name> ::= 
  <identifier>

<path pattern name> ::= 
  <identifier>

<parameter name> ::= 
  <dollar sign> <separated identifier>
  « WG3:RKE-031 »

<graph pattern variable> ::= 
  <element variable>
  | <path or subpath variable>

<element variable> ::= 
  <variable name>
  « WG3:RKE-031 »

```

```
<path or subpath variable> ::=  
  <path variable>  
  | <subpath variable>  
  
<path variable> ::=  
  <variable name>  
  
<subpath variable> ::=  
  <variable name>  
  
<binding variable name> ::=  
  <variable name>  
  
<variable name> ::=  
  <regular identifier>  
  
<identifier> ::=  
  <regular identifier>  
  | <delimited identifier>  
  
<separated identifier> ::=  
  <extended identifier>  
  | <delimited identifier>
```

Syntax Rules

None.

General Rules

- 1) An <object name> identifies an object.
- 2) A <graph name> identifies a graph.
« WG3:BER-040R3 deleted one GR »
- 3) A <binding table name> identifies a binding table.
- 4) A <value name> identifies a value.
- 5) A <procedure name> identifies a procedure.
- 6) A <function name> identifies a function.
- 7) A <label name> identifies a label.
- 8) A <property name> identifies a property of a graph, a node, or an edge.
- 9) A <field name> identifies a field of a record.
- 10) A <path pattern name> identifies a path that is matched by a path pattern.
- 11) A <parameter name> identifies a parameter.
- 12) A <binding variable name> identifies a binding variable.
- 13) An <element variable> identifies an element variable that may be bound to a list of graph elements.
« WG3:BER-031 »
- 14) A <path variable> identifies a path variable that is bound to a path binding that is matched by a <path pattern>.« WG3:RKE-031 »

- 15) A <subpath variable> identifies a subpath variable, which may be bound to a subpath of a path that is matched by a path pattern.

NOTE 153 — The subpath bound to a subpath variable may be empty the zero-node path.

Conformance Rules

None.

20.6 <token> and <separator>

Function

Specify lexical units (tokens and separators) that participate in the GQL language.

Format

```

<token> ::= 
    <non-delimiter token>
    | <delimiter token>

<non-delimiter token> ::= 
    <regular identifier>
    | <parameter name>
    | <key word>
    | <unsigned numeric literal>
    | <byte string literal>
    | <multiset alternation operator>

<non-delimited identifier> ::= 
    <regular identifier>
    | <extended identifier>

<regular identifier> ::= 
    <identifier start> [ <identifier extend>... ] 

    ** Editor's Note (number 262) **

The definition of <regular identifier>s should be extended to support more Unicode variants.
See Language Opportunity GQL-029.
```

```

<extended identifier> ::= 
    <identifier extend>...

<identifier start> ::= 
    !! See the Syntax Rules.

<identifier extend> ::= 
    !! See the Syntax Rules.

<key word> ::= 
    <reserved word>
    | <non-reserved word>

<reserved word> ::= 
    <case-insensitive reserved word>
    | endNode | inDegree | lTrim | outDegree | percentileCont | percentileDist | rTrim
    | startNode | stDev | stDevP | tail | toLower | toUpper
    « WG3:BER-082R1 »

<case-insensitive reserved word> ::= 
    ABS | ACOS | ADD | AGGREGATE | ALL | ALL_DIFFERENT | AND | ANY | ARRAY | AS | ASC
    | ASCENDING | ASIN | AT | ATAN | AVG
    « WG3:BER-085 »
    | BINARY | BIG | BIGINT | BOOL | BOOLEAN | BOTH | BY | BYTE_LENGTH | BYTES

```

20.6 <token> and <separator>

```

| CALL | CASE | CAST | CATALOG | CEIL | CEILING | CHARACTER | CHARACTER_LENGTH | CLEAR
| CLONE | CLOSE | COALESCE | COLLECT | COMMIT | CONSTRAINT | CONSTANT | CONSTRUCT | COPY
| COS | COSH | COT | COUNT | CURRENT_DATE | CURRENT_GRAPH | CURRENT_PROPERTY_GRAPH
| CURRENT_ROLE | CURRENT_SCHEMA | CURRENT_TIME | CURRENT_TIMESTAMP | CURRENT_USER | CREATE

| DATA | DATE | DATETIME | DEC | DECIMAL | DEFAULT | DEGREES | DELETE | DETACH | DESC
| DESCENDING | DIRECTORIES | DIRECTORY | DISTINCT | DO | DOUBLE | DROP | DURATION

| ELEMENT_ID | ELSE | ENDS | EMPTY_BINDING_TABLE | EMPTY_GRAPH
« WG3:BER-076 »
| EMPTY_PROPERTY_GRAPH | EMPTY_TABLE | EXCEPT | EXISTS | EXISTING | EXP

| FALSE | FILTER | FLOAT | FLOAT16 | FLOAT32 | FLOAT64 | FLOAT128 | FLOAT256
| FLOOR | FOR | FROM | FUNCTION | FUNCTIONS

| GQLSTATUS | GRANT | GROUP

| HAVING | HOME_GRAPH | HOME_PROPERTY_GRAPH | HOME_SCHEMA

| IN | INSERT | INT | INTEGER | INT8 | INTEGER8 | INT16 | INTEGER16 | INT32 | INTEGER32
| INT64 | INTEGER64 | INT128 | INTEGER128 | INT256 | INTEGER256
| INTERSECT | IF | IS

« WG3:BER-052R1 moved KEEP »
| LEADING | LEFT | LENGTH | LET | LIKE | LIKE_REGEX | LIMIT | LIST | LN
| LOCALDATETIME | LOCALTIME | LOCALTIMESTAMP | LOG | LOG10 | LOWER

| MANDATORY | MAP | MATCH | MERGE | MAX | MIN | MOD | MULTI | MULTIPLE
« WG3:BER-094R1 deleted one keyword »
| NEW | NOT | NORMALIZE | NOTHING | NULL | NULLS | NULLIF | NUMERIC

| OCCURRENCES_REGEX | OCTET_LENGTH | OF | OFFSET | ON | OPTIONAL | OR | ORDER | ORDERED
| OTHERWISE

| PARAMETER | PATH | PATHS | PARTITION | POSITION_REGEX | POWER | PRECISION | PROCEDURE
| PROCEDURES

« WG3:BER-076 »
| PRODUCT | PROJECT

| QUERIES | QUERY

| RADIANS | REAL | RECORD | RECORDS | REFERENCE | REMOVE | RENAME | REPLACE | REQUIRE
| RESET | RESULT | RETURN | REVOKE | RIGHT | ROLLBACK

| SAME | SCALAR | SCHEMA | SCHEMAS | SCHEMATA | SELECT | SESSION | SET | SKIP | SIGNED
| SIN

« WG3:BER-085 »
| SINGLE | SINH | SMALL | SMALLINT | SQRT | START | STARTS | STRING | SUBSTRING
| SUBSTRING_REGEX | SUM

| TAN | TANH | THEN | TIME | TIMESTAMP | TRAILING | TRANSLATE_REGEX | TRIM | TRUE
| TRUNCATE

« WG3:BER-085 »
| UBIGINT | UINT | UINT8 | UINT16 | UINT32 | UINT64 | UINT128 | UINT256 | UNION | UNIT
| UNIT_BINDING_TABLE | UNIT_TABLE | UNIQUE | UNNEST | UNKNOWN | UNSIGNED | UNWIND

« WG3:BER-085 »
| UPPER | USE | USMALLINT

| VALUE | VALUES | VARBINARY | VARCHAR

```

```
| WHEN | WHERE | WITH | WITHOUT
| XOR
| YIELD
| ZERO
| « WG3:BER-052R1 deleted one Editor's Note. »
<non-reserved word> ::=>
  <case-insensitive non-reserved word>

<case-insensitive non-reserved word> ::=>
  ACYCLIC

| BINDING
| « WG3:BER-032R3 »
| BINDINGS

| CLASS_ORIGIN | COMMAND_FUNCTION | COMMAND_FUNCTION_CODE | CONDITION_NUMBER | CONNECTING
| DESTINATION
| « WG3:BER-032R3 »
| DIFFERENT | DIRECTED

| EDGE | EDGES
| « WG3:BER-032R3 »
| ELEMENT | ELEMENTS

| FINAL | FIRST
| GRAPH | GRAPHS | GROUPS
| INDEX
| « WG3:BER-052R1 »
| KEEP

| LAST | LABEL | LABELED | LABELS
| MESSAGE_TEXT | MORE | MUTABLE
| NFC | NFD | NFKC | NFKD | NODE | NODES | NORMALIZED | NUMBER
| ONLY | ORDINALITY
| PATTERN | PATTERNS | PROPERTY | PROPERTIES
| READ | RELATIONSHIP | RELATIONSHIPS
| « WG3:BER-052R1 »
| REPEATABLE | RETURNED_GQLSTATUS

| SHORTEST | SIMPLE | SOURCE | SUBCLASS_ORIGIN
| TABLE | TABLES | TIES | TO | TRAIL | TRANSACTION | TYPE | TYPES
| UNDIRECTED
| VERTEX | VERTICES
| WALK | WRITE
| ZONE
```

20.6 <token> and <separator>

```

<multiset alternation operator> ::==
|+| !! <U+007C, U+002B, U+007C>

<delimiter token> ::=
  <GQL special character>
  | <bracket right arrow>
  | <bracket tilde right arrow>
  | <character string literal>
  | <concatenation operator>
  | <date string>
  | <datetime string>
  | <delimited identifier>
  | <double colon>
  | <double minus sign>
  | <double period>
  | <duration string>
  | <greater than operator>
  | <greater than or equals operator>
  | <left arrow>
  | <left arrow bracket>
  | <left arrow tilde>
  | <left arrow tilde bracket>
  | <left minus right>
  | <left minus slash>
  | <left tilde slash>
  | <less than operator>
  | <less than or equals operator>
  | <minus left bracket>
  | <minus slash>
  | <not equals operator>
  | <right arrow>
  | <right bracket minus>
  | <right bracket tilde>
  | <slash minus>
  | <slash minus right>
  | <slash tilde>
  | <slash tilde right>
  | <tilde left bracket>
  | <tilde right arrow>
  | <tilde slash>
  | <time string>

<bracket right arrow> ::=
]-> !! <U+005D, U+002D, U+003E>

<bracket tilde right arrow> ::=
]~> !! <U+005D, U+007E, U+003E>

<concatenation operator> ::=
|| !! <U+007C, U+007C>

<double colon> ::=
:: !! <U+003A, U+003A>

<double minus sign> ::=
-- !! <U+002D, U+002D>

<double period> ::=
.. !! <U+002E, U+002E>

<greater than operator> ::=
<right angle bracket>

```

```

<greater than or equals operator> ::==
    >= !! <U+003E, U+003D>

<left arrow> ::=
    <- !! <U+003C, U+002D>

<left arrow tilde> ::=
    <~ !! <U+003C, U+007E>

<left arrow bracket> ::=
    <-[ !! <U+003C, U+002D, U+005B>

<left arrow tilde bracket> ::=
    <~[ !! <U+003C, U+007E, U+005B>

<left minus right> ::=
    <-> !! <U+003C, U+002D, U+003E>

<left minus slash> ::=
    <- / !! <U+003C, U+002D, U+002F>

<left tilde slash> ::=
    <~/ !! <U+003C, U+007E, U+002F>

<less than operator> ::=
    <left angle bracket>

<less than or equals operator> ::=
    <= !! <U+003C, U+003D>

<minus left bracket> ::=
    <-[ !! <U+002D, U+005B>

<minus slash> ::=
    <- / !! <U+002D, U+002F>

<not equals operator> ::=
    <> !! <U+003C, U+003E>

<right arrow> ::=
    <-> !! <U+002D, U+003E>

<right bracket minus> ::=
    ]- !! <U+005D, U+002D>

<right bracket tilde> ::=
    ]~ !! <U+005D, U+007E>

<slash minus> ::=
    /- !! <U+002F, U+002D>

<slash minus right> ::=
    /-> !! <U+002F, U+002D, U+003E>

<slash tilde> ::=
    /~ !! <U+002F, U+007E>

<slash tilde right> ::=
    /~> !! <U+002F, U+007E, U+003E>

<tilde left bracket> ::=
    ~[ !! <U+007E, U+005B>

<tilde right arrow> ::=
    ~> !! <U+007E, U+003E>

```

20.6 <token> and <separator>

```

<tilde slash> ::= ~ / !! <U+007E, U+002F>

<delimited identifier> ::= <double quoted character sequence>
| <unbroken accent quoted character sequence>

<double solidus> ::= // /!<U+002F, U+002F>

<separator> ::= { <comment> | <whitespace> }...

<whitespace> ::= !! See the Syntax Rules.

<comment> ::= <simple comment>
| <bracketed comment>

<simple comment> ::= <simple comment introducer> [ <simple comment character>... ] <newline>

<simple comment introducer> ::= <double solidus>
| <double minus sign>

<simple comment character> ::= !! See the Syntax Rules.

<bracketed comment> ::= <bracketed comment introducer>
| <bracketed comment contents>
| <bracketed comment terminator>

<bracketed comment introducer> ::= /* /!<U+002F, U+002A>

<bracketed comment terminator> ::= */ /!<U+002A, U+002F>

<bracketed comment contents> ::= !! See the Syntax Rules.

<newline> ::= !! See the Syntax Rules.

<edge synonym> ::= EDGE | RELATIONSHIP
« WG3:BER-032R3 »

<edges synonym> ::= EDGES | RELATIONSHIPS

<node synonym> ::= NODE | VERTEX

```

Syntax Rules

- 1) An <identifier start> is any character with the Unicode property ID_Start as defined by UAX31-D1 Default Identifier Syntax in **Unicode Standard Annex #31**.

NOTE 154 — The characters in ID_Start are those in the Unicode General Category classes “Lu”, “Ll”, “Lt”, “Lm”, “Lo”, and “Nl” together with those with the Unicode property Other_ID_Start but none of which have the Unicode properties Pattern_Syntax or Pattern_White_Space.

- 2) An <identifier extend> is any character with the Unicode property ID_Continue as defined by UAX31-D1 Default Identifier Syntax in [Unicode Standard Annex #31](#).

NOTE 155 — The characters in ID_Continue are those in ID_Start together with those in the Unicode General Category classes “Mn”, “Mc”, “Nd”, and “Pc” together with those with the Unicode property Other_ID_Continue but none of which have the Unicode properties Pattern_Syntax or Pattern_White_Space.

- 3) The *representative form RF* of a <non-delimited identifier> NDI is the character string comprising the sequence of characters contained in NDI. RF shall not be the zero-length character string.
- 4) The maximum length in characters of the representative form of a <non-delimited identifier> shall be $2^{14} - 1 = 16383$.

NOTE 156 — This maximum length is modified by [Conformance Rule 1](#).

- 5) The *representative form RF* of a <delimited identifier> DI is the character string specified by the <single quoted character sequence>, the <double quoted character sequence>, or the <unbroken accent quoted character sequence> contained in DI. RF shall not be the zero-length character string.
- 6) The maximum length in characters of the representative form of a <delimited identifier> shall be $2^{14} - 1 = 16383$.

NOTE 157 — This maximum length is modified by [Conformance Rule 2](#).

- 7) <whitespace> is any consecutive sequence of Unicode characters with the property White_Space.
- 8) <simple comment character> is any Unicode character that is not a <newline>.
- 9) <bracketed comment contents> is any character string of Unicode characters that does not contain <bracketed comment terminator>.
- 10) <newline> is the implementation-defined ([ID084](#)) end-of-line indicator.

NOTE 159 — <newline> is typically represented by U+000A (“Line Feed”) and/or U+000D (“Carriage Return”); however, this representation is not required by this document.

- 11) A <token>, other than a <byte string literal>, <character string literal>, or a <delimited identifier>, shall not contain a <separator>.
- 12) Any <token> may be followed by a <separator>. A <non-delimiter token> shall be followed by a <delimiter token> or a <separator>.

NOTE 160 — If the Format does not allow a <non-delimiter token> to be followed by a <delimiter token>, then that <non-delimiter token> shall be followed by a <separator>.

- 13) GQL text containing one or more instances of <simple comment> is equivalent to the same GQL text with each <simple comment> replaced with <newline>.
- 14) GQL text containing one or more instances of <bracketed comment> is equivalent to the same GQL text with each <bracketed comment> BC replaced with,

Case:

- a) If BC contains a <newline>, then <newline>.
- b) Otherwise, <space>.

- 15) For every <non-delimited identifier> *NDI* there is exactly one corresponding *case-normal form CNF*. *CNF* is a character string derived from the representative form *RF* of *NDI* as follows:

Let *n* be the number of characters in *RF*. For *i* ranging from 1 (one) to *n*, the *i*-th character *M_i* of *RF* is transliterated into the corresponding character or characters of *CNF* as follows

Case:

- a) If *M_i* is a lower-case character or a title case character for which an equivalent upper-case sequence *U* is defined by Unicode, then let *j* be the number of characters in *U*; the next *j* characters of *CNF* are *U*.
- b) Otherwise, the next character of *CNF* is *M_i*.

NOTE 161 — Any lower-case letters for which there are no upper-case equivalents are left in their lower-case form.

NOTE 162 — The case-normal form of a <non-delimited identifier> is used in excluding <reserved word>s.

- 16) The case-normal form of a <non-delimited identifier> shall not be equal, according to the comparison rules in Subclause 18.3, “<comparison predicate>”, to any <case-insensitive reserved word> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter or letters), treated as a <character string literal>.

NOTE 163 — It is the intention that no <key word> specified in this document or revisions thereto shall end with an <underscore>.

- 17) Two <non-delimited identifier>s are equivalent if their representative forms compare equally according to the comparison rules in Subclause 18.3, “<comparison predicate>”.
- 18) A <non-delimited identifier> and a <delimited identifier> are equivalent if their representative forms compare equally according to the comparison rules in Subclause 18.3, “<comparison predicate>”.
- 19) Two <delimited identifier>s are equivalent if their representative forms compare equally according to the comparison rules in Subclause 18.3, “<comparison predicate>”.
- 20) For the purposes of identifying <case-insensitive reserved word>s, any <simple Latin lower-case letter> contained in a candidate <key word> shall be effectively treated as the corresponding <simple Latin upper-case letter>.
- 21) For the purposes of identifying <case-insensitive non-reserved word>s, any <simple Latin lower-case letter> contained in a candidate <key word> shall be effectively treated as the corresponding <simple Latin upper-case letter>.

General Rules

None.

Conformance Rules

- 1) Without Feature GB00, “Long identifiers”, in conforming GQL language, the maximum length in characters of the representative form of a <non-delimited identifier> shall be $2^7 - 1 = 127$.
- 2) Without Feature GB00, “Long identifiers”, in conforming GQL language, the maximum length in characters of the representative form of a <delimited identifier> shall be $2^7 - 1 = 127$.
- 3) Without Feature GB01, “Double minus sign comments”, conforming GQL language shall not contain a <simple comment> introduced with a <double minus sign>.

- 4) Without Feature GB02, “Double solidus comments”, conforming GQL language shall not contain a <simple comment> introduced with a <double solidus>.

20.7 <GQL terminal character>

Function

Define the terminal symbols of the GQL language.

Format

```

<GQL terminal character> ::==
  <GQL language character>
  | <other language character>

<GQL language character> ::==
  <simple Latin letter>
  | <digit>
  | <GQL special character>

<simple Latin letter> ::==
  <simple Latin lower-case letter>
  | <simple Latin upper-case letter>

<simple Latin lower-case letter> ::=
  a | b | c | d | e | f | g | h | i | j | k | l | m | n | o
  | p | q | r | s | t | u | v | w | x | y | z

<simple Latin upper-case letter> ::=
  A | B | C | D | E | F | G | H | I | J | K | L | M | N | O
  | P | Q | R | S | T | U | V | W | X | Y | Z

<hex digit> ::=
  <standard digit> | A | B | C | D | E | F | a | b | c | d | e | f

<digit> ::=
  <standard digit>
  | <other digit>

<standard digit> ::=
  <octal digit> | 8 | 9

<octal digit> ::=
  <binary digit> | 2 | 3 | 4 | 5 | 6 | 7

<binary digit> ::=
  0 | 1

<other digit> ::=
  !! See the Syntax Rules.

<GQL special character> ::=
  <space>
  | <campersand>
  | <asterisk>
  | <colon>
  | <equals operator>
  | <comma>
  | <dollar sign>
  | <double quote>
  | <exclamation mark>
  | <grave accent>
  | <right angle bracket>
  | <left brace>

```

```
| <left bracket>
| <left paren>
| <left angle bracket>
| <minus sign>
| <period>
| <plus sign>
| <question mark>
| <quote>
| <reverse solidus>
| <right brace>
| <right bracket>
| <right paren>
| <semicolon>
| <solidus>
| <underscore>
| <vertical bar>
| <percent>
| <circumflex>
| <tilde>

<space> ::= !!"U+0020

<ampersand> ::= & !! "U+0026

<asterisk> ::= *

<circumflex> ::= ^ !! "U+005E

<colon> ::= :
: !! "U+003A

<comma> ::= ,
, !! "U+002C

<dollar sign> ::= $
$ !! "U+0024

<double quote> ::= "
" !! "U+0022

<equals operator> ::= =
= !! "U+003D

<exclamation mark> ::= !
! !! "U+0021

<right angle bracket> ::= >
> !! "U+003E

<grave accent> ::= `
` !! "U+0060

<left brace> ::= {
{ !! "U+007B

<left bracket> ::= [
[ !! "U+005B

<left paren> ::= (
( !! "U+0028
```

```

<left angle bracket> ::==
  < !! U+003C

<minus sign> ::==
  - !! U+002D

<percent> ::==
  % !! U+0025

<period> ::==
  . !! U+002E

<plus sign> ::==
  + !! U+002B

<question mark> ::==
  ? !! U+003F

<quote> ::==
  ' !! U+0027

<reverse solidus> ::==
  \ !! U+005C

<right brace> ::==
  } !! U+007D

<right bracket> ::==
  ] !! U+005D

<right paren> ::==
  ) !! U+0029

<:semicolon> ::==
  ; !! U+003B

<solidus> ::==
  / !! U+002F

<tilde> ::==
  ~ !! U+007E

<underscore> ::==
  _ !! U+005F

<vertical bar> ::==
  | !! U+007C

<other language character> ::=
  !! See the Syntax Rules.

```

Syntax Rules

- 1) <other language character> is any Unicode character not contained in <GQL language character>.
- 2) <other digit> is any Unicode character in the Unicode General Category class “Nd” not contained in <standard digit>.
- 3) The <hex digit>s a, b, c, d, e, and f have respectively the same values as the <hex digit>s A, B, C, D, E, and F.

General Rules

- 1) There is a one-to-one correspondence between the symbols contained in <simple Latin upper-case letter> and the symbols contained in <simple Latin lower-case letter> such that, for every i , the symbol defined as the i -th alternative for <simple Latin upper-case letter> corresponds to the symbol defined as the i -th alternative for <simple Latin lower-case letter>.

Conformance Rules

None.

21 Additional common rules

** Editor's Note (number 263) **

SQL has Subclause 9.16, "Potential sources of non-determinism" in SQL/Foundation, which is modified by Subclause 9.1, "Potential sources of non-determinism" in SQL/PGQ as a result of WG3:W04-009R1. GQL currently has no equivalent Subclause. See Language Opportunity [GQL-011](#).

21.1 Satisfaction of a <label expression> by a label set

Subclause Signature

```
"Satisfaction of a <label expression> by a label set" [Syntax Rules] (
    Parameter: "LABEL EXPRESSION",
    Parameter: "LABEL SET"
) Returns: "TRUTH VALUE"
```

Function

Determine if a label set satisfies a <label expression>.

Syntax Rules

- 1) Let $LEXP$ be the *LABEL EXPRESSION* and let LS be the *LABEL SET* in an application of the Syntax Rules of this Subclause. The result of the application of this Subclause is TV , which is returned as *TRUTH VALUE*.
- 2) A label set LS satisfies a <label expression> LE according to the following recursive definition:
 - a) If LE is a <label name> $L2$, then $L2$ is a member of LS .
 - b) If LE is a <wildcard label>, then LS is non-empty.

NOTE 164 — This condition is always true; every label set is non-empty. The rule is written this way in case empty label sets are permitted in the future, or for guidance to a GQL-implementation that supports graph elements with no labels.

« [WG3:BER-080](#) »

- c) If LE is a <parenthesized label expression> PLE , then let $LE2$ be the <label expression> simply contained in PLE ; LS satisfies $LE2$.
 - d) If LE is a <label negation>, then let LP be the <label primary> simply contained in LE ; LS does not satisfy LP .
 - e) If LE is a <label conjunction>, then let $L1$ be the <label term> and let $L2$ be the <label factor> simply contained in LE ; LS satisfies $L1$ and LS satisfies $L2$.
 - f) If LE is a <label disjunction>, then let $L1$ be the <label expression> and let $L2$ be the <label term> simply contained in LE ; LS satisfies $L1$ or LS satisfies $L2$.
- 3) TV is

Case:

21.1 Satisfaction of a <label expression> by a label set

- a) If LS satisfies $LEXP$, then *True*.
- b) Otherwise, *False*.

General Rules

None.

Conformance Rules

None.

21.2 Machinery for graph pattern matching

Subclause Signature

```
"Machinery for graph pattern matching" [General Rules] (
  Parameter: "PURE PROPERTY GRAPH",
  Parameter: "PATH PATTERN LIST"
) Returns: "MACHINERY"
```

Function

Define the infrastructure (alphabet, mappings and related definitions) used in graph pattern matching.

Syntax Rules

None.

General Rules

- 1) Let PG be the *PURE PROPERTY GRAPH* and let PPL be the *PATH PATTERN LIST* in an application of the General Rules of this Subclause. The result of the application of this Subclause is $MACH$, which is returned as *MACHINERY*.
 « WG3:BER-031 »
- 2) Let SVV be the set of names of node variables declared in PPL at the same depth of graph pattern matching, and let SEV be the set of names of edge variables declared in PPL at the same depth of graph pattern matching.
- 3) For each subpath variable SPV declared in PPL at the same depth of graph pattern matching, let $SPVBEGIN$ and $SPVEND$ be two distinct <identifier>s that are distinct from every <identifier> in $SVV \cup SEV$ and from every <identifier> created by this rule. $SPVBEGIN$ is the *begin subpath symbol* and $SPVEND$ is the *end subpath symbol* associated with SPV . Let SPS be the set of every subpath symbol.
- 4) Let ' $\langle\rangle$ ' and ' $\langle\rangle$ ' be mutually distinct <identifier>s that are distinct from every <identifier> in $SVV \cup SEV \cup SPS$. These are, respectively, the *anonymous node symbol* and the *anonymous edge symbol*. Let SAS be the set of anonymous symbols.
- 5) Let NPP be the number of <parenthesized path pattern expression>s contained in PPL at the same depth of graph pattern matching. Let $PPPE_1, \dots, PPPE_{NPP}$ be an enumeration of the <parenthesized path pattern expression>s contained in PPL at the same depth of graph pattern matching. For every i , $1 \leq i \leq NPP$, i is the *bracket index* of $PPPE_i$.
- 6) Let ' $[_1, \dots, [_{NPP}],]_1, \dots,]_{NPP}$ ' be $2 * NPP$ <identifier>s that are mutually distinct, and distinct from every member of $SVV \cup SEV \cup SPS \cup SAS \cup SEPS$ and from every graph element of PG . These are called *bracket symbols*. For every bracket symbols ' $_j$ ' or ' $]_j$ ', j is the *bracket index* of the bracket symbol. There are two bracket symbols for each bracket index j between 1 (one) and NPP , corresponding to the <parenthesized path pattern expression>s $PPPE_j$. Let SBS be the set of bracket symbols.
- 7) Let GX be the set whose members are the graph elements of PG , the subpath symbols, and the bracket symbols.
 « WG3:BER-096 »

- 8) Let ABC be $SPS \cup SBS \cup SVV \cup SEV \cup SAS$. ABC is the *alphabet*. The members of ABC are *symbols*.
- 9) A *word* is a string of elements of ABC .
- 10) An *elementary binding* is a pair (LET, GE) where LET is a member of ABC and GE is a member of GX , such that:

Case:

- a) If LET is a bracket symbol, then $LET = GE$. In this case, the elementary binding is a *bracket symbol binding*. If LET is a start bracket symbol, then the elementary binding is a *start bracket symbol binding*; otherwise, it is an *end bracket symbol binding*. The bracket index of LET is the bracket index of the bracket symbol binding.

« WG3:BER-031 »

- b) If LET is a subpath symbol, then $LET = GE$. In this case, the elementary binding is a *subpath symbol binding*.
- c) If LET is the name of a node variable or the anonymous node symbol, then GE is a node. In this case the elementary binding is a *node symbol binding*.
- d) If LET is the name of an edge variable or the anonymous edge symbol, then GE is an edge. In this case, the elementary binding is an *edge symbol binding*.
- 11) If $EB = (LET, GE)$ is an elementary binding, then EB is an *elementary binding of LET*, and EB binds LET to GE .

NOTE 165 — An elementary binding is a mapping of a symbol (an <identifier>) to a member of GX ; it is not the binding of a graph pattern variable. In particular, if LET is the name of an element variable EV , there may be more than one elementary binding of LET in a multi-path binding. In a consistent path binding (defined subsequently), two elementary bindings of LET must bind to the same graph element in contexts in which LET is exposed as unconditional singleton; otherwise elementary bindings of LET are independent of one another, and may bind to more than one graph element. Similarly, references to EV are context-dependent, and may resolve to a list which is a proper subset of all the graph elements bound to LET by elementary bindings. Resolution of <element reference>s is performed by the General Rules of Subclause 21.5, “Applying bindings to evaluate an expression”.

**** Editor's Note (number 264) ****

The following item from BER-031 was not included. A *compressed binding* is a pair (LOV, GE) where LOV is a list of names of primary variables and GE is a graph element.

- 12) When an elementary binding (LET, GE) binds LET to a graph element GE during the evaluation of an <element pattern where clause>, temporarily add a new field whose field name is LET and whose field value is a graph element reference value to GE to the current working record until the evaluation has finished.

**** Editor's Note (number 265) ****

This General Rule was added in this document and is not included in SQL/PGQ. See Possible Problem [GQL-164](#).

- 13) A *path binding* is a sequence of zero or more elementary bindings, $B = (LET_1, E_1), \dots, (LET_N, E_N)$. Given a path binding B :

NOTE 166 — The definition of path recognizes the zero-node path (one having no graph elements) as a path; therefore the definition of path binding also allows a sequence of zero elementary bindings. For example, a quantifier may iterate 0 (zero) times, resulting in an empty path binding. The result of a <path pattern>, on the other hand, is required to be non-empty because of a Syntax Rule that its minimum node count shall be at least 1 (one).

- a) The *word* of B is the sequence LET_1, \dots, LET_N .
- b) The *annotated path* of B is the sequence E_1, \dots, E_N .

21.2 Machinery for graph pattern matching

NOTE 167 — If the path binding is consistent (defined subsequently), the annotated path of B contains within it a path that matches the word of B , plus mark-up with bracket symbols and subpath symbols indicating how to interpret the path as a match to the word.

« WG3:BER-054 »

- c) The *compressed path binding* CPB of B is obtained from B as follows:

- i) Let CPB be a copy of B .

« WG3:BER-031 »

- ii) All subpath symbol bindings and all bracket symbol bindings are deleted from CPB .
 - iii) Each maximal subsequence MS of CPB comprising one or more consecutive node bindings is replaced by a single compressed binding, whose components are the following:

« WG3:BER-031 »

- 1) The first component is a list of the names of node variables in the first component of the node bindings of MS .

NOTE 168 — This list is empty if only the anonymous node symbol is bound in MS .

- 2) The second component is the node that is bound by the first node binding in MS .

NOTE 169 — For consistent path bindings, consecutive node bindings will bind the same node.

« WG3:BER-031 »

- iv) In each edge binding EB of CPB , the first component is replaced by a list of zero or one name of an edge variable, retaining the name of the edge variable in the first component of EB , if any.

- d) The *extracted path* XP of B is obtained from the compressed path binding of B as the sequence of the second components of the compressed bindings of CPB .

« WG3:BER-054 »

- 14) Let $REDUCE$ be a function that maps path bindings to path bindings, determined as follows:

- a) Let $PBIN$ be a path binding.

- b) Let $PBOUT$ be a copy of $PBIN$.

- c) All bracket bindings are removed from $PBOUT$.

- d) The following steps are performed on $PBOUT$ repeatedly until no more anonymous node bindings can be removed:

- i) If there are two adjacent anonymous node bindings, then the second is removed.

- ii) If there is a binding of a node variable adjacent to an anonymous node binding, then the anonymous node binding is removed.

- e) $REDUCE(PBIN)$ is $PBOUT$.

- 15) A *reduced path binding* RPB is obtained from a path binding PB as $RPB = REDUCE(PB)$.

« WG3:BER-054 deleted one rule. »

- 16) When a path binding $PB = (LET_1, E_1), \dots (LET_N, E_N)$ for a path or subpath variable $POSPV$ is *applied* in the scope of some element E during the application of a General Rule $RULE$, then:

- a) Let RPB be the reduced path binding obtained from PB .

- b) Let F be a new field whose field name is $POSPV$ and whose field value is a path value constructed from the graph element reference values corresponding to the graph elements of the extracted path of RPB in the same order.
- c) Add F temporarily to the current working record until the application of $RULE$ has completed.
- d) For every singleton graph pattern variable $SGPV$ that is visible in the scope of E and that is bound by PB to a single graph element SGE , temporarily add a new field whose field name is $SGPV$ and whose field value is a graph element reference value to SGE to the current working record until the application of $RULE$ has completed.
- e) For every group graph pattern variable $GGPV$ that is visible in the scope of E and that is bound by PB to a set of multiple graph elements MGE , temporarily add a new field whose field name is $GGPV$ and whose field value is a list of graph element reference values to the graph elements of MGE to the current working record until the application of $RULE$ has completed.
- f) For every subpath variable binding $(SPVBEGIN, SPVBEGIN) (LET_j, E_j), \dots (LET_k, E_k) (SPVEND, SPVEND)$, $1 \leq j \leq k \leq N$, for some subpath variable SV from S whose subpath begin symbol is $SPVBEGIN$ and whose subpath end symbol is $SPVEND$ and that is contained in PB :
 - i) Let $RSPB$ be the reduced path binding obtained from the subpath binding $SPB = (LET_j, E_j), \dots, (LET_k, E_k)$ for SV .
 - ii) Let SF be a new field whose field name is SV and whose field value is a path value constructed from the graph element reference values corresponding to the graph elements of the extracted path of $RSPB$ in the same order.
 - iii) Add SF temporarily to the current working record until the application of $RULE$ has completed.

**** Editor's Note (number 266) ****

It needs to be decided, if further nesting of subpath variables is to be exposed, e.g., via additional properties on path values. This would add support for the representation of nested solutions. See Language Opportunity [GQL-194](#).

**** Editor's Note (number 267) ****

The entire preceding General Rule was added in this document and is not included in SQL/PGQ. See Possible Problem [GQL-164](#).

- 17) In a path binding, two node bindings are *separable* if there is an edge binding between them.
- 18) A path binding B is *consistent* if the following conditions are true:
 - a) The extracted path XP of B is a path of PG ; that is, either XP is the zero-node path or XP begins with a node, alternates between nodes and edges, each edge in XP connects the node before and after it, and XP ends with a node.
 - b) For every two node bindings (LET_1, E_1) and (LET_2, E_2) that are not separable, $E_1 = E_2$.
 « WG3:BER-031 »
 NOTE 170 — If there are only bracket symbol or subpath symbol bindings between two node bindings, the node bindings must bind to the same node.
 « WG3:BER-080 »
 - c) For every edge binding $EB = (LET, GE)$, let (LET_{left}, GE_{left}) be the last node binding to the left of EB and let $(LET_{right}, GE_{right})$ be the first node binding to the right of EB .

Case:

21.2 Machinery for graph pattern matching

- i) If GE is an undirected edge, then GE is an edge connecting GE_{left} and GE_{right} .
- ii) If GE is a directed edge, then either GE_{left} is the source node and GE_{right} is the destination node of GE , or GE_{right} is the source node and GE_{left} is the destination node of GE .

NOTE 171 — The directionality constraint of the edge binding is fully checked during the generation of the regular language for <path concatenation>.

- d) For every start bracket symbol binding SBS contained in B , let j be the bracket index of SBS .

« Editorial: Align wording with SQL/PGQ »

All of the following are true:

- i) There is an end bracket symbol binding contained in B and following SBS whose bracket index is j . Let EBS be the first such end bracket symbol binding. Let PPS be the explicit or implicit <path mode prefix> simply contained in the <parenthesized path pattern expression> whose bracket index is j . Let PM be the <path mode> contained in PPS .

- ii) Case:

- 1) If PM is TRAIL, then there is no pair of edge bindings at two different positions between SBS and EBS that bind the same edge.

NOTE 172 — This definition does not take note of the symbols that are bound, only the edges. It is a violation of the TRAIL <path mode> if the symbols in the pair of distinct edge bindings are both anonymous edge symbols, are both edge variables (whether the same or different), or one is the anonymous edge symbol and the other is an edge variable.

- 2) If PM is SIMPLE, then no two separable node bindings between SBS and EBS bind the same node, except that the first and last node binding between SBS and EBS may bind the same node.

NOTE 173 — This definition does not take note of the symbols that are bound, only the nodes. It is a violation of the SIMPLE <path mode> if the symbols in the pair of separable node bindings are both anonymous node symbols, are both node variables (whether the same or different), or one is the anonymous node symbol and the other is a node variable. However, the first and last node binding between SBS and EBS may bind the same node without violating the SIMPLE <path mode>.

- 3) If PM is ACYCLIC, then no two separable node bindings between SBS and EBS bind the same node.

NOTE 174 — This definition does not take note of the symbols that are bound, only the nodes. It is a violation of the ACYCLIC <path mode> if the symbols in the pair of separable node bindings are both anonymous node symbols, both are node variables (whether the same or different), or one is the anonymous node symbol and the other is a node variable.

NOTE 175 — The <path mode> WALK imposes no constraints on the extracted path.

« WG3:BER-032R3 deleted one Editor's Note. »

- 19) A *multi-path binding* is an n-tuple (PB_1, \dots, PB_n) for some positive integer n such that each PB_i is a path binding.

- 20) If S and T are sets of strings, let $S \cdot T$ be the set of strings formed by concatenating an element of S followed by an element of T , that is, $S \cdot T = \{ s t \mid s \text{ is an element of } S, t \text{ is an element of } T \}$.

NOTE 176 — The \cdot operator will be used to concatenate words (strings of symbols) and path bindings (strings of elementary bindings).

- 21) If S is a set of strings, then let S^0 be the set whose only element is the string of length 0 (zero), and for each non-negative integer n , let S^{n+1} be $S^n \cdot S$. Let S^* be the union of S^n for every non-negative integer n .

NOTE 177 — If S is not empty, then S^* is an infinite set; however, a finite result for every <graph pattern> is assured by the syntactic requirement that every <quantified path primary> is bounded, contained in a restrictive <parenthesized path pattern expression> or contained in a selective <path pattern>.

« WG3:BER-054 moved one rule. »

- 22) *REDUCE* is extended to multi-path bindings as follows: If $MPB = (PB_1, \dots, PB_n)$ is a multi-path binding, then $REDUCE(MPB) = (REDUCE(PB_1), \dots, REDUCE(PB_n))$.
- 23) Let *MACH* be a data structure comprising the following:
 - a) *ABC*, the alphabet, formed as the disjoint union of the following:
 - i) *SVV*, the set of names of node variables.
 - ii) *SEV*, the set of names of edge variables.
 - iii) *SPS*, the set of subpath symbols.
 - iv) *SAS*, the set of anonymous symbols.
 - v) *SBS*, the set of bracket symbols.
 - b) *REDUCE*, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.
- 24) *MACH* is returned as the *MACHINERY*.

Conformance Rules

None.

21.3 Evaluation of a <path pattern expression>

Subclause Signature

"Evaluation of a <path pattern expression>" [General Rules] (

- Parameter: "PURE PROPERTY GRAPH",
- Parameter: "PATH PATTERN LIST",
- Parameter: "MACHINERY",
- Parameter: "SPECIFIC BNF INSTANCE"

) Returns: "SET OF MATCHES"

Function

Evaluate a <path pattern expression>.

Syntax Rules

None.

General Rules

- 1) Let PG be the *PURE PROPERTY GRAPH*, let PPL be the *PATH PATTERN LIST*, let $MACH$ be the *MACHINERY*, and let SBI be the *SPECIFIC BNF INSTANCE* in an application of the General Rules of this Subclause. The result of the application of this Subclause is SM , which is returned as *SET OF MATCHES*.

NOTE 178 — The parameters have the following meanings:

SPECIFIC BNF INSTANCE: the specific instance of a BNF non-terminal to be evaluated.

SET OF MATCHES: returns the set of partial matches to the *SPECIFIC BNF INSTANCE*.

- 2) The following components of $MACH$ are identified:

- a) ABC , the alphabet, formed as the disjoint union of the following:

«WG3:BER-031»

- i) SVV , the set of names of node variables.
- ii) SEV , the set of names of edge variables.
- iii) SPS , the set of subpath symbols.
- iv) SAS , the set of anonymous symbols.
- v) SBS , the set of bracket symbols.

- b) $REDUCE$, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.

- 3) For every instance BNT of a BNF non-terminal that is a <path pattern expression>, <path term>, <path pattern union>, <path factor>, <path concatenation>, <path primary>, <quantified path primary>, <questioned path primary>, <element pattern>, or <parenthesized path pattern expression> equal to or contained in SBI at the same depth of graph pattern matching, the following are defined by simultaneous recursion:

21.3 Evaluation of a <path pattern expression>

- the *regular language of BNT*, denoted $RL(BNT)$, defined as a set of words over ABC ; and
- the *set of local matches to BNT*, denoted $SLM(BNT)$, defined as a set of path bindings.

NOTE 179 — $SLM(BNT)$ is consistent except when concatenating an edge pattern prior to concatenating the following node pattern. Restrictive path modes are enforced when generating $SLM(BNT)$ for the <parenthesized path pattern expression> that declares the path mode.

NOTE 180 — $RL(BNT)$ and $SLM(BNT)$ may be infinite sets if BNT contains an effectively unbounded quantifier. Every effectively unbounded quantifier is required to be contained in a selective <path pattern>; the potentially infinite set of local matches is subsequently reduced to a finite set by the General Rules of Subclause 21.4, “Evaluation of a selective <path pattern>”.

NOTE 181 — The BNF non-terminals are listed above in the “top down” order of appearance in the Format of Subclause 16.7, “<path pattern expression>”; the definitions in the following subrules treat the same BNF non-terminals in “bottom up” order.

Case:

- a) If BNT is a <parenthesized path pattern expression>, then let PPE be the <path pattern expression> immediately contained in BNT and let j be the bracket index of BNT .

NOTE 182 — “Bracket index” is defined in Subclause 21.2, “Machinery for graph pattern matching”.

i) Case:

- 1) If BNT simply contains a <subpath variable declaration> SVD , then let SV be the subpath variable declared by SVD . Let BSV be the begin subpath symbol associated with SV and let ESV be the end subpath symbol associated with SV . Let $BSVBINDING$ be (BSV, BSV) and let $ESVBINDING$ be (ESV, ESV) .
- 2) Otherwise, let BSV , ESV , $BSVBINDING$ and $ESVBINDING$ be the empty string.

ii) $RL(BNT)$ is $\{ [j] \cdot \{ BSV \} \cdot RL(PPE) \cdot \{ ESV \} \cdot]_j \}$

iii) Let $STPB$ be $SLM(PPE)$.

Let $SLMMAYBE$ be $\{ ([j], [j]) \cdot \{ BSVBINDING \} \cdot STPB \cdot \{ ESVBINDING \} \cdot ([j], [j]) \}$

$SLM(BNT)$ is the set of every path binding in $SLMMAYBE$ that is consistent.

« WG3:BER-031 »

NOTE 183 — That is, the words in $RL(BNT)$ are formed by surrounding the words of $RL(PPE)$ by the bracket symbols $[j]$ and $]_j$. If BNT contains a subpath variable declaration, then the words of $RL(BNT)$ are also surrounded by the begin and end subpath symbol associated with that subpath variable. Similarly, the path bindings in $SLMMAYBE$ are formed by surrounding the path bindings with bracket bindings and, if there is a subpath declaration, with the corresponding begin and end subpath bindings. Eliminating inconsistent bindings from $SLMMAYBE$ to get $SLM(BNT)$ has the effect of enforcing restrictive path modes.

- b) If BNT is an <element pattern> EP , then:

i) Case:

« WG3:BER-031 »

- 1) If EP declares an element variable EV , then let EPI be the name of EV .
 - 2) If EP is a <node pattern>, then let EPI be ‘ \emptyset ’, the anonymous node symbol.
 - 3) If EP is an <edge pattern>, then let EPI be ‘ $-$ ’, the anonymous edge symbol.
- ii) $RL(BNT)$ is $\{ EPI \}$, the set whose sole member is EPI .
 - iii) $SLM(BNT)$ is the set of every elementary binding (EPI, GE) such that:
 - 1) If EP is a <node pattern>, then GE is a node.

21.3 Evaluation of a <path pattern expression>

- 2) If EP is an <edge pattern>, then GE is an edge.
- 3) If EP simply contains a <label expression> LE , then the value of TV is True when the Syntax Rules of Subclause 21.1, "Satisfaction of a <label expression> by a label set", are applied with LE as **LABEL EXPRESSION** and the label set of GE as **LABEL SET**; let TV be the **TRUTH VALUE** returned from the application of those Syntax Rules.
- c) If BNT is a <quantified path primary>, then:
 - i) Let PP be the <path primary> immediately contained in BNT . As a result of the transformations in the Syntax Rules, PP is a <parenthesized path pattern expression>. Let R be $RL(PP)$ and let S be $SLM(PP)$.
 - ii) Let GQ be the <general quantifier> immediately contained in BNT .
 - iii) Let LB be the value of the <lower bound> contained in GQ .
 - iv) Case:

« WG3:BER-080 »

- 1) If GQ contains an <upper bound>, then let UB be the value of the <upper bound>.
 - A) $RL(BNT)$ is $R^{LB} \cup R^{LB+1} \cup \dots \cup R^{UB-1} \cup R^{UB}$.
 - B) Let $TOOMUCH$ be $S^{LB} \cup S^{LB+1} \cup \dots \cup S^{UB-1} \cup S^{UB}$.
 - C) $SLM(BNT)$ is the set of those path bindings in $TOOMUCH$ that are consistent.
- 2) Otherwise,
 - A) $RL(BNT)$ is $R^{LB} \cdot R^*$.
 - B) Let $WAYTOOMUCH$ be $S^{LB} \cdot S^*$.
 - C) $SLM(BNT)$ is the set of those path bindings in $WAYTOOMUCH$ that are consistent.
- d) If BNT is a <questioned path primary>, then:
 - i) Let PP be the <path primary> immediately contained in BNT . As a result of the transformations in the Syntax Rules, PP is a <parenthesized path pattern expression>. Let R be $RL(PP)$ and let S be $SLM(PP)$.
 - ii) $RL(BNT)$ is $R^0 \cup R$.
 - iii) $SLM(BNT)$ is $S^0 \cup S$.

« WG3:BER-080 »

- e) If BNT is a <path primary>, then let $BNT2$ be the <element pattern> or <parenthesized path pattern expression> that is immediately contained in BNT .
 - i) $RL(BNT)$ is $RL(BNT2)$.
 - ii) $SLM(BNT)$ is $SLM(BNT2)$.
- f) If BNT is a <path concatenation>, then:
 - i) Let PST be the <path term> and let PC be the <path factor> that are immediately contained in BNT .

21.3 Evaluation of a <path pattern expression>

ii) $RL(BNT)$ is $RL(PST) \cdot RL(PC)$.

iii) Case:

1) If PC is an <edge pattern>, then $SLM(BNT)$ is $SLM(PST) \cdot SLM(PC)$

NOTE 184 — If PC is an <edge pattern>, then there is a <node pattern> to its right, which will be concatenated in a subsequent iteration of this recursion; consistency, including the directionality constraint implied by the <edge pattern>, will be checked at that point.

2) If PC is a <node pattern>, and the last <element pattern> EP of PST is an <edge pattern> then:

NOTE 185 — By transformations in the Syntax Rules of Subclause 16.7, “<path pattern expression>”, an edge binding is always immediately preceded and followed by a node binding. The current rule handles the situation in which the edge has been bound as the last elementary binding of PST , and therefore PC is the node binding that immediately follows the edge binding. Also note that the Syntax Rules have transformed every <abbreviated edge pattern> to a <full edge pattern>.

A) Let $SLMCONCAT$ be $SLM(PST) \cdot SLM(PC)$.

B) For each path binding PB contained in $SLMCONCAT$,

I) Let GE_{right} be the node that is bound in the last elementary binding of PB .

II) Let GE be the edge that is bound in the penultimate elementary binding of PB .

III) Let GE_{left} be the node that is bound in the antepenultimate elementary binding of PB .

C) Let the propositions L , U , and R be determined as follows:

I) Proposition L is true if GE is a directed edge, GE_{left} is the destination node of GE and GE_{right} is the source node of GE .

II) Proposition U is true if GE is an undirected edge, and GE_{left} and GE_{right} are the nodes connected by GE .

III) Proposition R is true if GE is a directed edge, GE_{left} is the source node of GE and GE_{right} is the destination node of GE .

D) Let the *directionality constraint* of EP be

Case:

I) If EP is a <full edge pointing left>, then proposition L is true.

II) If EP is a <full edge undirected>, then proposition U is true.

III) If EP is a <full edge pointing right>, then proposition R is true.

IV) If EP is a <full edge left or undirected>, then proposition L , or proposition U , is true.

V) If EP is a <full edge undirected or right>, then proposition U , or proposition R , is true.

VI) If EP is a <full edge left or right>, then proposition L , or proposition R , is true.

21.3 Evaluation of a <path pattern expression>

- VII) If EP is a <full edge any direction>, then at least one of proposition L , proposition U , or proposition R , is true.
- E) $SLM(BNT)$ is the set of those path bindings of $SLMCONCAT$ that are consistent and satisfy the directionality constraint of EP .
- 3) Otherwise, $SLM(BNT)$ is the set of those path bindings in $SLM(PST) \cdot SLM(PC)$ that are consistent.

**** Editor's Note (number 268) ****

It may be possible to enforce implicit joins of unconditional singletons exposed by a <path concatenation> as part of the GRs for <path concatenation>. This was discussed in an SQL/PGQ ad hoc meeting on September 8, 2020. It was decided not to attempt that change as part of WG3:W04-009R1, leaving it as a future possibility. See Language Opportunity [GQL-044](#).

« WG3:BER-080 »

- g) If BNT is a <path factor>, then let $BNT2$ be the <path primary>, <quantified path primary> or <questioned path primary> immediately contained in BNT .
- i) $RL(BNT)$ is $RL(BNT2)$.
 - ii) $SLM(BNT)$ is $SLM(BNT2)$.

« WG3:BER-080 »

- h) If BNT is a <path pattern union>, then let NMA be the number of <path term>s immediately contained in BNT . Let PMO_1, \dots, PMO_{NMA} be these <path term>s.
- i) $RL(BNT)$ is $RL(PMO_1) \cup RL(PMO_2) \cup \dots \cup RL(PMO_{NMA})$.
 - ii) $SLM(BNT)$ is $SLM(PMO_1) \cup SLM(PMO_2) \cup \dots \cup SLM(PMO_{NMA})$.

« WG3:BER-080 »

- i) If BNT is <path term>, then let $BNT2$ be the <path factor> or <path concatenation> immediately contained in BNT .
- i) $RL(BNT)$ is $RL(BNT2)$.
- ii) $SLM(BNT)$ is $SLM(BNT2)$.

« WG3:BER-080 »

- j) If BNT is a <path pattern expression>, then let $BNT2$ be the <path term> or <path pattern union> immediately contained in BNT .

NOTE 186 — <path multiset alternation> is transformed into <path pattern union> in the Syntax Rules and therefore it is not considered separately here.

- i) $RL(BNT)$ is $RL(BNT2)$.
- ii) $SLM(BNT)$ is $SLM(BNT2)$.

4) Let SM be $SLM(SBI)$.

5) SM is returned as the *SET OF MATCHES*.

Conformance Rules

None.

21.4 Evaluation of a selective <path pattern>

Subclause Signature

```
"Evaluation of a selective <path pattern>" [General Rules] (
    Parameter: "PURE PROPERTY GRAPH",
    Parameter: "PATH PATTERN LIST",
    Parameter: "MACHINERY",
    Parameter: "SELECTIVE PATH PATTERN",
    Parameter: "INPUT SET OF LOCAL MATCHES"
) Returns: "OUTPUT SET OF LOCAL MATCHES"
```

Function

Evaluate a <path pattern> with a selective <path search prefix>.

Syntax Rules

None.

General Rules

- 1) Let *PG* be the *PURE PROPERTY GRAPH*, let *PPL* be the *PATH PATTERN LIST*, let *MACH* be the *MACHINERY*, let *SEL* be the *SELECTIVE PATH PATTERN*, and let *INSLM* be the *INPUT SET OF LOCAL MATCHES* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *OUTSLM*, which is returned as *OUTPUT SET OF LOCAL MATCHES*.

NOTE 187 — The parameters have the following meaning:

SEL: a selective <path pattern>.

INPUT SET OF LOCAL MATCHES: the (potentially infinite) set of local matches to *SEL* generated by Subclause 21.3, "Evaluation of a <path pattern expression>".

OUTPUT SET OF LOCAL MATCHES: returns a finite subset of *INPUT SET OF LOCAL MATCHES* selected according to the <path search prefix> of *SEL*.

- 2) It is implementation-defined (IA013) whether the General Rules of this Subclause are terminated if an exception condition is raised. If a GQL-implementation defines that it terminates execution because of an exception condition, it is implementation-dependent (UA006) which of the members of *CANDIDATES* (defined subsequently) are actually probed to establish whether they might raise an exception.

NOTE 188 — *CANDIDATES* is potentially an infinite set, but there are algorithms to enumerate this set so as to satisfy the selection criterion of the selective <path pattern> without testing all candidate solutions. Even if the GQL-implementation defines that it terminates when an exception condition is encountered on a particular candidate solution, the order of enumerating the candidates may be implementation-dependent, and a candidate solution that might raise an exception may never be tested.

- 3) The following components of *MACH* are identified:

- a) *ABC*, the alphabet, formed as the disjoint union of the following:

« WG3:BER-031 »

- i) *SVV*, the set of names of node variables.
- ii) *SEV*, the set of names of edge variables.

21.4 Evaluation of a selective <path pattern>

- iii) *SPS*, the set of subpath symbols.
 - iv) *SAS*, the set of anonymous symbols.
 - v) *SBS*, the set of bracket symbols.
- b) *REDUCE*, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.
- 4) Let *NP* be the number of <path pattern>s in *PPL*.
- 5) *SEL* is a selective <path pattern>. Let *j* be the bracket index of the <parenthesized path pattern expression> simply contained in *SEL*.
- NOTE 189 — “Bracket index” is defined in Subclause 21.2, “Machinery for graph pattern matching”.
- NOTE 190 — By a syntactic transformation in Subclause 16.5, “<graph pattern>”, this <parenthesized path pattern expression> is the entire content of *SEL* except possibly the declaration of a path variable.
- 6) Let *PSP* be the <path search prefix> simply contained in *SEL*.
- 7) Let *N* be the value of the <number of paths> or the <number of groups> specified in *PSP*. If *N* is not a positive integer, then an exception condition is raised: *data exception — invalid number of paths or groups* (22G0F) and no further General Rules of this Subclause are applied.
- 8) Let *p* be such that *SEL* is the *p*-th <path pattern> of *PPL*.
- 9) Let *CANDIDATES* be the set of every path binding *PBX* in *INSLM* such that the following conditions are true:

« WG3:BER-031 »

- a) For every unconditional singleton <element variable> *EV* exposed by *SEL*, *EV* is bound to a unique graph element by the elementary bindings of *EV* contained in *PBX*.
- NOTE 191 — Anonymous symbols are not <element variable>s; there is no requirement that two anonymous symbols bind to the same graph element.
- b) For every <parenthesized path pattern expression> *PPPE* equal to or contained in *SEL*, let *i* be the bracket index of *PPPE*, and let *[i]* and *]i* be the bracket symbols associated with *PPPE*. A *binding* of *PPPE* is a substring of *PBX* that begins with the bracket binding (*[i]*, *[i]*) and ends with the next bracket binding (*]i*, *]i*).

NOTE 192 — “Bracket index” is defined in Subclause 21.2, “Machinery for graph pattern matching”.

For every binding *BPPPE* of *PPPE* contained in *PBX*, the following are true:

« WG3:BER-031 »

- i) For every <element variable> *EV* that is exposed as an unconditional singleton by *PPPE*, *EV* is bound to a unique graph element by the elementary bindings of *EV* contained in *BPPPE*.
- NOTE 193 — Anonymous symbols are not <element variable>s; there is no requirement that two anonymous symbols bind to the same graph element.
- ii) If *PPPE* contains a <parenthesized path pattern where clause> *PPPWC*, then the value of *V* is *True* when the General Rules of Subclause 21.5, “Applying bindings to evaluate an expression”, are applied with *PPL* as *GRAPH PATTERN*, the <search condition> simply contained in *PPPWC* as *EXPRESSION*, *MACH* as *MACHINERY*, *PBX* as *MULTI-PATH BINDING*, and a reference to *BPPPE* as a subset of *PBX* as *REFERENCE TO LOCAL CONTEXT*; let *V* be the *VALUE* returned from the application of those General Rules.

NOTE 194 — This is the juncture at which an exception condition might be raised. It is implementation-defined whether to terminate if an exception condition is raised. The order of enumerating the

members of *CANDIDATES* is implementation-dependent, and there is no requirement that a GQL-implementation test all candidate solutions, which may be an infinite set in any case.

- 10) Each path binding *PBX* of *CANDIDATES* is replaced by *REDUCE(PBX)*.
- 11) Redundant duplicate path bindings are removed from *CANDIDATES*.
- 12) *CANDIDATES* is partitioned as follows: For every path binding *PBX* in *CANDIDATES*, the partition of *PBX* is the set of every path binding *PBY* in *CANDIDATES* such that the first and last node bindings of *PBX* bind the same nodes as the first and last node bindings, respectively, of *PBY*.
- 13) Each partition *PART* of *CANDIDATES* is modified as follows

Case:

 - a) If *PSP* is an <any path search>, then

Case:

 - i) If the number of path bindings in *PART* is *N* or less, then the entire partition *PART* is retained.
 - ii) Otherwise, it is implementation-dependent (UA005) which *N* path bindings of *PART* are retained.
 - b) If *PSP* is a <shortest path search>, then

Case:

 - i) If *PSP* is a <counted shortest path search>, then

Case:

 - 1) If the number of path bindings in *PART* is *N* or less, then the entire partition *PART* is retained.
 - 2) Otherwise, the path bindings of *PART* are sorted in increasing order of number of edges; the order of path bindings that have the same number of edges is implementation-dependent (US005). The first *N* path bindings in *PART* are retained.
 - ii) If *PSP* is <counted shortest group search>, then the path bindings in *PART* are grouped, with each group comprising those path bindings having the same number of edges. The groups are ordered in increasing order by the number of edges.

Case:

 - 1) If the number of groups in *PART* is *N* or less, then the entire partition *PART* is retained.
 - 2) Otherwise, the path bindings comprising the first *N* groups of *PART* are retained.
- 14) Let *OUTSLM* be the set of path bindings retained in *CANDIDATES* after the preceding modifications to its partitions.
- 15) *OUTSLM* is returned as *OUTPUT SET OF LOCAL MATCHES*.

Conformance Rules

None.

21.5 Applying bindings to evaluate an expression

Subclause Signature

"Applying bindings to evaluate an expression" [General Rules] (

- Parameter: "GRAPH PATTERN",
- Parameter: "EXPRESSION",
- Parameter: "MACHINERY",
- Parameter: "MULTI-PATH BINDING",
- Parameter: "REFERENCE TO LOCAL CONTEXT"

) Returns: "VALUE"

Function

« WG3:BER-031 »

Evaluate a <value expression> or <search condition> using the bindings of graph pattern variables determined by a local context within a multi-path binding.

Syntax Rules

None.

General Rules

- 1) Let *GP* be the *GRAPH PATTERN*, let *EXP* be the *EXPRESSION*, let *MACH* be the *MACHINERY*, let *MPB* be the *MULTI-PATH BINDING*, and let *RTLC* be the *REFERENCE TO LOCAL CONTEXT* in an application of the General Rules of this Subclause. The result of the application of this Subclause is *V*, which is returned as *VALUE*.

NOTE 195 — The parameters have the following meaning:

GRAPH PATTERN: a <graph pattern>.

EXPRESSION: a <value expression> or <search condition>.

MULTI-PATH BINDING: multi-path binding to the <graph pattern> in which to evaluate the expression.

REFERENCE TO LOCAL CONTEXT: an indication of a subset of the multi-path binding, the local context. Group bindings are confined to the local context; singleton bindings may look outside the local context. The local context is passed "by reference" in order to correctly evaluate non-local singletons. For example, given the pattern:

((A) -> ((B) -> (C) WHERE A.X = B.X+C.X) -> (D)) {2}

then A.X makes a non-local reference to element variable A. A word of this pattern will repeat the outer <parenthesized path pattern expression> twice, requiring two evaluations of the WHERE clause. Each evaluation of the WHERE clause must locate the appropriate non-local reference to A. The bindings to the inner <parenthesized path pattern expression>, if passed "by value", might not be enough information to determine the appropriate binding to the outer <parenthesized path pattern expression>.

- 2) The following components of *MACH* are identified:

- a) *ABC*, the alphabet, formed as the disjoint union of the following:

« WG3:BER-031 »

- i) *SVV*, the set of names of node variables.

21.5 Applying bindings to evaluate an expression

- ii) SEV , the set of names of edge variables.
 - iii) SPS , the set of subpath symbols.
 - iv) SAS , the set of anonymous symbols.
 - v) SBS , the set of bracket symbols.
 - b) $REDUCE$, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.
- 3) Let LC be the subset of MPB that is indicated by $RTLC$.
- 4) For each $\langle\text{element reference}\rangle ER$ contained in EXP at the same depth of graph pattern matching,
« WG3:BER-031 »

**** Editor's Note (number 269) ****

Reference to iterator variables from BER-031 not included.

- a) Let DEG be the degree of reference of ER .
- b) Let EV be the $\langle\text{element variable}\rangle$ of ER .
- c) Case:
 - i) If LC is equal to MPB , then let $SPACE$ be MPB .

NOTE 196 — That is, the search space is the entire multi-path binding. This case arises in two circumstances: 1) the evaluation of a $\langle\text{parenthesized path pattern where clause}\rangle$ in the outermost $\langle\text{parenthesized path pattern expression}\rangle$ of a selective $\langle\text{path pattern}\rangle$; 2) the evaluation of a $\langle\text{graph pattern where clause}\rangle$.

- ii) Otherwise, let $LCBI$ be the bracket index of the first bracket symbol in LC . Let $LCPPPE$ be the $\langle\text{parenthesized path pattern expression}\rangle$ contained in GP whose bracket index is $LCBI$. Let PP be the $\langle\text{path pattern}\rangle$ containing $LCPPPE$.

NOTE 197 — “Bracket index” is defined in Subclause 21.2, “Machinery for graph pattern matching”.

Case:

- 1) If EV is not declared by PP then let $SPACE$ be MPB .

NOTE 198 — In this case, EV is declared in some other $\langle\text{path pattern}\rangle$ than the one that contains ER . ER is a non-local reference, therefore DEG is singleton, and EV must be exposed as a singleton by the $\langle\text{path pattern}\rangle$ (s) that declare it.

- 2) If EV is declared by $LCPPPE$ then let $SPACE$ be LC .

NOTE 199 — In this case, EV is declared locally to $LCPPPE$ and the binding(s) to ER can be found by searching the local context LC .

- 3) Otherwise, let $DEFPPPE$ be the innermost $\langle\text{parenthesized path pattern expression}\rangle$ that declares EV and that contains $LCPPPE$. Let BI be the bracket index of $DEFPPPE$. Let $'[BI'$ and $']BI'$ be the bracket symbols whose bracket index is BI . Let $SPACE$ be the smallest substring of MPB containing LC and beginning with $'[BI'$ and ending with $']BI'$.

NOTE 200 — In this case, EV is declared in some outer scope containing $LCPPPE$, and the binding of ER , if any, is found by searching the innermost scope that declares EV . ER is a non-local reference, therefore DEG is singleton, and EV must be exposed as a singleton by $DEFPPPE$.

NOTE 201 — “Bracket index” is defined in Subclause 21.2, “Machinery for graph pattern matching”.

- d) Case:

21.5 Applying bindings to evaluate an expression

- i) If DEG is singleton, then

Case:

« WG3:BER-031 »

- 1) If there is an elementary binding EB of EV in $SPACE$, then let LOE be a list with a single graph element, the graph element that is bound to EV by EB .

NOTE 202 — Even if EV is bound multiple times in $SPACE$ (expressing an equijoin on EV), the list has only one graph element.

- 2) Otherwise, let LOE be the empty list.

NOTE 203 — This case can only arise if DEG is conditional singleton.

- ii) If DEG is group, then

Case:

- 1) If $SPACE$ does not contain an elementary binding of EV , then let LOE be an empty list.

« WG3:BER-031 »

- 2) Otherwise, let LOE be the list of the graph elements that are bound to EV by elementary bindings in the order that they occur in $SPACE$, scanning $SPACE$ from left to right, and retaining duplicates.

**** Editor's Note (number 270) ****

The bindings of a group reference flatten nested lists. This may be acceptable for SQL aggregates, which have no support for nested groupings, but may be inadequate to fully capture the semantics of a group reference in a graph pattern. WG3:MMX-035r2 section 4.1 "Desynchronized lists" pointed out a problem with reducing group variables to lists: two lists may be interleaved, but the reduction to separate lists can lose this information. The example given is

```
( (A:Person) -[:SPOUSE]-> ()
  | (B:Person) -[:FRIEND]-> () ) {3}
```

A solution may find matches to A and B in any order. With separate lists of matches of A and B, it will not be easy to reconstruct the precise sequence of interleaved matches to A and B.

A similar problem can arise with nested quantifiers. WG3:MMX-035r2 section 4.2 "Nested quantifiers" gives this example:

```
( (C1:CORP) ( -[:TRANSFERS] -> (B:BANK) ) *
  -[:TRANSFERS]-> (C2:CORP) ) *
```

With this pattern, there can be 0 (zero) or more bindings to B between any two consecutive bindings to C1 and C2. With just independent lists of matches to C1, B and C2, it will not be easy to determine which bindings to B lie between which bindings to C1 and C2.

- e) LOE is the list of graph elements bound to ER .

NOTE 204 — This list is used by the General Rules of those Subclauses that evaluate ER , for example, Subclause 19.19, "*property reference*".

- 5) Let V be the result of EXP .
- 6) For each *<element reference>* ER contained in EXP at the same depth of graph pattern matching, the list of graph elements bound to ER is destroyed.

Conformance Rules

None.

21.6 Determination of identical values

Subclause Signature

```
"Determination of identical values" [General Rules] (
    Parameter: "FIRST VALUE",
    Parameter: "SECOND VALUE"
)
```

Function

Determine whether two values are identical.

Syntax Rules

None.

General Rules

- 1) Let $V1$ be the *FIRST VALUE* and let $V2$ be the *SECOND VALUE* in an application of the General Rules of this Subclause.

NOTE 205 — This Subclause is invoked implicitly wherever the word *identical* is used of two values.

- 2) Whether $V1$ is identical to $V2$ is determined as follows

Case:

**** Editor's Note (number 271) ****

The way in which material values are determined to be identical and the definition of comparable may require further adjustment (as per the discussion of RKE-048).

- a) If $V1$ and $V2$ are both the null value, then $V1$ is identical to $V2$.
- b) If $V1$ is the null value and $V2$ is not the null value, or if $V1$ is not the null value and $V2$ is the null value, then $V1$ is not identical to $V2$.
- c) If $V1$ and $V2$ are of comparable types, then $V1$ is identical to $V2$ if and only if $V1$ is not distinct from $V2$.
- d) Otherwise, $V1$ is not identical to $V2$.

21.7 Determination of distinct values

Subclause Signature

```
"Determination of distinct values" [General Rules] (
    Parameter: "FIRST VALUE",
    Parameter: "SECOND VALUE"
)
```

Function

Determine whether two values are distinct.

Syntax Rules

None.

General Rules

- 1) Let $V1$ be the *FIRST VALUE* and let $V2$ be the *SECOND VALUE* in an application of the General Rules of this Subclause.

NOTE 206 — This Subclause is invoked implicitly wherever the word *distinct* is used of two values.

- 2) Whether $V1$ is distinct from $V2$ is determined as follows

Case:

**** Editor's Note (number 272) ****

The way in which material values are determined to be distinct may require further adjustment (as per the discussion of RKE-048).

- a) If $V1$ and $V2$ are both the null value, then $V1$ is not distinct from $V2$.
- b) If $V1$ is the null value and $V2$ is not the null value, or if $V1$ is not the null value and $V2$ is the null value, then $V1$ is distinct from $V2$.
- c) If $V1$ and $V2$ are of a predefined value type, then $V1$ is distinct from $V2$ if and only if $V1$ is not equal to $V2$.
- d) If $V1$ and $V2$ are of a record type, then $V1$ is distinct from $V2$ if and only if the field values of at least one of the respective fields of $V1$ and $V2$ with the same field name are distinct.
- e) Otherwise, $V1$ is not distinct from $V2$.

21.8 Equality operations

Function

Specify the prohibitions and restrictions by value type on operations that involve testing for equality.

Syntax Rules

- 1) An *equality operation* is any of the following:

**** Editor's Note (number 273) ****

Others to be added as the predicates or expressions that need them are also added. If none are added before DIS the list needs to be collapsed. See [Possible Problem GQL-189](#).

**** Editor's Note (number 274) ****

Consider explicit support for additional collations other than UCS BASIC and UNICODE (e.g., case insensitive variants, use of alternative collation tables). See [Language Opportunity GQL-012](#).

- a) A <comparison predicate> that specifies <equals operator> or <not equals operator>.

General Rules

None.

Conformance Rules

None.

21.9 Ordering operations

Function

Specify the prohibitions and restrictions by value type on operations that involve ordering of data.

Syntax Rules

- 1) An *ordering operation* is any of the following:

**** Editor's Note (number 275) ****

Others to be added as the predicates or expressions that need them are also added. See Possible Problem [GQL-188](#).

**** Editor's Note (number 276) ****

Consider explicit support for additional collations other than UCS BASIC and UNICODE (e.g., case insensitive variants, use of alternative collation tables). See Language Opportunity [GQL-012](#).

- a) A <comparison predicate> that does not specify <equals operator> or <not equals operator>.
- b) A <sort specification list>.
- c) An <aggregate function> that specifies MAX or MIN.

General Rules

**** Editor's Note (number 277) ****

Comparison is currently outlined as part of Subclause 4.12, "Data types". This topic requires further discussion, detail, and alignment and should eventually be considered for being moved here. See Possible Problem [GQL-174](#).

None.

Conformance Rules

None.

21.10 Collation determination

Subclause Signature

"Collation determination" [Syntax Rules] () Returns: "COLL"

Function

Specify rules for determining the collation to be used in the comparison of character strings.

Syntax Rules

- 1) The Syntax Rules of this Subclause are applied without any symbolic arguments. The result of the application of this Subclause is the collation to be used, which is returned as *COLL*.
- 2) *COLL* is defined as the implementation-defined (ID022) choice of one of the following:
 - a) UCS_BASIC.
 - b) UNICODE.
 - c) An implementation-defined (ID022) custom collation.

General Rules

None.

Conformance Rules

None.

21.11 Result of value type combinations

Subclause Signature

"Result of value type combinations" [General Rules] (

Parameter: "DTSET"

) Returns: "RESTYPE"

**** Editor's Note (number 278) ****

WG3:W05-020 suggests that discussion of this Subclause is needed to establish consensus. See Possible Problem [GQL-087](#).

Function

« Consequence of WG3:BER-094R1 »

Specify the declared type of the result of certain combinations of values of compatible value types, such as <case expression>s, or <list value expression>s.

Syntax Rules

**** Editor's Note (number 279) ****

[BER-019] and [BER-094R1] established that rules on types should be specified in Syntax Rules. This Subclause has to be adjusted accordingly, as well as the wording of every rule that calls the subclause. See Possible Problem [GQL-299](#).

None.

General Rules

- 1) Let *IDTS* be the *DTSET* in an application of the General Rules of this Subclause. The result of the application of this Subclause is the declared type of the result, which is returned as *RESTYPE*.
- 2) Let *DTS* be the set of value types in *IDTS*.
- 3) Case:
 - a) If any of the value types in *DTS* is a character string type, then if any value type in *DTS* is not a character string type, then an exception condition is raised: *data exception — invalid value type (22G03)*.
 - b) If any of the value types in *DTS* is a byte string type, then if any value types in *DTS* are not byte string types an exception condition is raised: *data exception — invalid value type (22G03)*.
 - c) If any value type in *DTS* is numeric, then:
 - i) If any value type in *DTS* is not numeric, then an exception condition is raised: *data exception — invalid value type (22G03)*.
 - ii) The declared type of the result is
Case:
 - 1) If any value type in *DTS* is approximate numeric, then approximate numeric with implementation-defined ([ID037](#)) precision.

21.11 Result of value type combinations

- 2) Otherwise, exact numeric with implementation-defined (IL016) precision and with scale equal to the maximum of the scales of the value types in *DTS*.

d) If any value type in *DTS* is a DATETIME or LOCALDATETIME value type, then:

Case:

- i) If any value type in *DTS* is not a DATETIME or LOCALDATETIME value type, then an exception condition is raised: *data exception — invalid value type (22G03)*.
- ii) If any value type in *DTS* is DATETIME, then the declared type of the result is DATETIME.
- iii) Otherwise, the declared type of the result is LOCALDATETIME.

e) If any value type in *DTS* is a TIME or LOCALTIME value type, then:

Case:

- i) If any value type in *DTS* is not a TIME or LOCALTIME value type then an exception condition is raised: *data exception — invalid value type (22G03)*.
- ii) If any value type in *DTS* is TIME, then the declared type of the result is TIME.
- iii) Otherwise, the declared type of the result is LOCALTIME.

f) If any value type in *DTS* is DURATION, then:

Case:

- i) If any value type in *DTS* is not DURATION then an exception condition is raised: *data exception — invalid value type (22G03)*.
- ii) Otherwise, the declared type of the result is DURATION.

g) If any value type in *DTS* is a Boolean value, then:

Case:

- i) if any value type in *DTS* is not a Boolean value then an exception condition is raised: *data exception — invalid value type (22G03)*.
- ii) Otherwise, the declared type of the result is the Boolean type.

h) If any value type in *DTS* is a list type, then:

Case:

- i) If any value type in *DTS* is not a list type then an exception condition is raised: *data exception — invalid value type (22G03)*.
- ii) Otherwise, the declared type of the result is the list type with element value type *ETR*, where *ETR* is the value type resulting from the application of this Subclause to the set of element types of the list types of *DTS*.

« WG3:BER-094R1 deleted two subrules »

- i) If any value type in *DTS* is a map type, then:

Case:

- i) If any value type in *DTS* is not a map type then an exception condition is raised: *data exception — invalid value type (22G03)*.
- ii) Otherwise, declared type of the result is the map type with key element value type *KETR*, where *KETR* is the value type resulting from the application of this Subclause to

the set of key element types of the map types of *DTS* and value element value type *VETR*, where *VETR* is the value type resulting from the application of this Subclause to the set of value element types of the map types of *DTS*.

Conformance Rules

None.

« WG3:BER-094R1 deleted one Subclause »

22 Diagnostics

General Rules

- 1) When a condition is raised the following map, DIAGNOSTICS, is created with diagnostic information.

```
MAP { COMMAND_FUNCTION: CF,
      COMMAND_FUNCTION_CODE: CFC,
      NUMBER: N,
      CURRENT_SCHEMA: CS,
      HOME_GRAPH: HG,
      CURRENT_GRAPH: CG
}
```

where:

- a) *CF* is a string identifying the command executed. [Table 6, “Command codes”](#), specifies the identifier of the commands.
- b) *CFC* is an integer identifying the code of the GQL-statement executed. [Table 6, “Command codes”](#), identifies the code of the commands. Positive values are reserved for commands defined by this document. Negative values are reserved for implementation-defined commands.
- c) *N* is the number chained GQL-status objects.
- d) *CS* is the value of CURRENT_SCHEMA.
- e) *HG* is the value of HOME_GRAPH.
- f) *CG* is the value of CURRENT_GRAPH.
- g) If the GQLSTATUS corresponds to *syntax error or access rule violation — invalid reference (42002)*, then the following map element is added to DIAGNOSTICS:

```
INVALID_REFERENCE: R
```

where *R* is the identifier of the reference that caused the exception condition to be raised.

Table 6 — Command codes

Command	Identifier	Code
<session set schema clause>	SESSION SET SCHEMA	1 (one)
<session set graph clause>	SESSION SET GRAPH	2
<session set time zone clause>	SESSION SET TIME ZONE	3
<session set parameter clause>	SESSION SET PARAMETER	4
<session clear command>	SESSION CLEAR	5
<session close command>	SESSION CLOSE	6
<session remove command>	SESSION REMOVE	7

Command	Identifier	Code
<start transaction command>	START TRANSACTION	8
<rollback command>	ROLLBACK	9
<commit command>	COMMIT	10
<create graph statement>	CREATE GRAPH STATEMENT	11
<create graph type statement>	CREATE GRAPH TYPE STATEMENT	12
<drop graph statement>	DROP GRAPH STATEMENT	18
<drop graph type statement>	DROP GRAPH TYPE STATEMENT	19
<match statement>	MATCH STATEMENT	26
<call query statement>	CALL QUERY STATEMENT	27
<call procedure statement>	CALL PROCEDURE STATEMENT	28
<do statement>	DO STATEMENT	29
<insert statement>	INSERT STATEMENT	30
« WG3:BER-050 deleted one row »		
<set statement>	SET STATEMENT	32
<remove statement>	REMOVE STATEMENT	33
<delete statement>	DELETE STATEMENT	34
<optional statement>	OPTIONAL STATEMENT	35
<mandatory statement>	MANDATORY STATEMENT	36
<let statement>	LET STATEMENT	37
<for statement>	FOR STATEMENT	38
<aggregate statement>	AGGREGATE STATEMENT	39
<filter statement>	FILTER STATEMENT	40
<order by and page statement>	ORDER BY AND PAGE STATEMENT	41
Implementation-defined statements	An implementation-defined character string value different from the value associated with any other command	x ¹
Unrecognized statements	A zero-length character string	0 (zero)

¹ An implementation-defined negative number different from the value associated with any other statement in the GQL language.

23 Status codes

23.1 GQLSTATUS

A character string returned as GQLSTATUS is the concatenation of a 2-character class code followed by a 3-character subclass code, each restricted to <standard digit>s and <simple Latin upper-case letter>s, respectively. [Table 7, “GQLSTATUS class and subclass codes”](#), categorizes and specifies the class code for each condition and the subclass code or codes for each subcondition defined by this document.

Class codes that begin with one of the <standard digit>s '0', '1', '2', '3', or '4' or one of the <simple Latin upper-case letter>s 'A', 'B', 'C', 'D', 'E', 'F', 'G', or 'H' are returned only for conditions defined in this document or some other International Standard. The range of such class codes is called *standard-defined classes*. Some such class codes are reserved for use by specific International Standards, as specified elsewhere in this Clause. Subclass codes associated with such classes that also begin with one of those 13 characters are returned only for conditions defined in this document or some other International Standard. The range of such subclass codes is called *standard-defined subclasses*. Subclass codes associated with such classes that begin with one of the <standard digit>s '5', '6', '7', '8', or '9' or one of the <simple Latin upper-case letter>s 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', or 'Z' are reserved for implementation-defined conditions and are called *implementation-defined subclasses*.

Class codes that begin with one of the <standard digit>s '5', '6', '7', '8', or '9' or one of the <simple Latin upper-case letter>s 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', or 'Z' are reserved for implementation-defined exception conditions and are called *implementation-defined classes*. All subclass codes except '000', which means *no subclass*, associated with such classes are reserved for implementation-defined conditions and are called *implementation-defined subclasses*. An implementation-defined completion condition shall be indicated by returning an implementation-defined subclass in conjunction with one of the classes *successful completion (00000)*, *warning (01000)*, or *no data (02000)*.

If a subclass code is not specified for a condition, then either subclass '000' or an implementation-defined subclass is returned.

The “Category” column has the following meanings: 'S' means that the class code given corresponds to a completion condition that indicates successful completion and; 'W' means that the class code given corresponds to a completion condition that indicates successful completion but with a warning; 'N' means that the class code given corresponds to a completion condition that indicates a no-data situation; 'X' means that the class code given corresponds to an exception condition.

Table 7 — GQLSTATUS class and subclass codes

Category	Condition	Class	Subcondition	Subclass
S	<i>successful completion</i>	00	(<i>no subclass</i>)	000
W	<i>warning</i>	01	(<i>no subclass</i>)	000
			<i>string data, right truncation</i>	004
			<i>graph already exists</i>	G01
			<i>graph type already exists</i>	G02
			<i>graph does not exist</i>	G03

Category	Condition	Class	Subcondition	Subclass
			<i>graph type does not exist</i>	G04
N	<i>no data</i>	02	(no subclass)	000
X	<i>connection exception</i>	08	(no subclass)	000
			<i>transaction resolution unknown</i>	007
X	<i>data exception</i>	22	(no subclass)	000
			<i>string data, right truncation</i>	001
			<i>numeric value out of range</i>	003
			<i>null value not allowed</i>	004
			<i>invalid datetime format</i>	007
			<i>invalid time zone displacement value</i>	009
			<i>substring error</i>	011
			<i>division by zero</i>	012
			<i>invalid character value for cast</i>	018
			<i>invalid argument for natural logarithm</i>	01E
			<i>invalid argument for power function</i>	01F
			<i>trim error</i>	027
			<i>negative limit value</i>	G02
			<i>invalid value type</i>	G03
			<i>values not comparable</i>	G04
			<i>invalid datetime function map key</i>	G05
			<i>invalid datetime function map value</i>	G06
			<i>invalid duration function map key</i>	G07
			<i>list data, right truncation</i>	G0B
			<i>list element error</i>	G0C
			<i>empty binding table returned</i>	G0D

Category	Condition	Class	Subcondition	Subclass
			<i>invalid number of paths or groups</i>	G0F
			<i>invalid duration format</i>	G0H
			<i>multiple assignments to a graph element property</i>	G0M
X	<i>invalid transaction state</i>	25	(no subclass)	000
			<i>active GQL-transaction</i>	G01
			<i>catalog and data statement mixing not supported</i>	G02
X	<i>invalid transaction termination</i>	2D	(no subclass)	000
X	<i>transaction rollback</i>	40	(no subclass)	000
			<i>integrity constraint violation</i>	002
			<i>statement completion unknown</i>	003
X	<i>syntax error or access rule violation</i>	42	(no subclass)	000
			<i>invalid syntax</i>	001
			<i>invalid reference</i>	002
X	<i>procedure call error</i>	G0	(no subclass)	000
			<i>incorrect number of arguments</i>	001
			<i>invalid parameter type</i>	002
			<i>invalid result type</i>	003
X	<i>dependent object error</i>	G1	(no subclass)	000
			<i>edges still exist</i>	001
X	<i>graph type conformance error</i>	G2	(no subclass)	000
X	<i>graph type definition error</i>	G3	(no subclass)	000
			<i>endpoint node type not defined</i>	001

24 Conformance

24.1 Introduction to conformance

Conformance may be claimed by a graph, a GQL-program, or a GQL-implementation.

24.2 Minimum conformance

Minimum conformance is defined as meeting the requirements of the data model and all syntax and semantics not explicitly identified as belonging to an optional feature.

A claim of minimum conformance shall also include:

« WG3:BER-030 »

- 1) A claim of conformance to:

« WG3:BER-032R3 »

- Feature G001, “Repeatable-elements match mode”.
- Feature G008, “Graph Pattern Where”.
- Feature G034, “Path concatenation”.
- Feature G040, “Vertex pattern”.
- Feature G042, “Basic Full Edge Patterns”.
- Feature G070, “Label expression: Label disjunction”.
- Feature G073, “Label expression: Individual label name”.
- Feature G090, “Property reference”.

- 2) A claim of conformance to at least one of:

- Feature GA02, “Empty node label set support”
- Feature GA03, “Singleton node label set support”

- 3) A claim of conformance to at least one of:

- Feature GA05, “Empty edge label set support”
- Feature GA06, “Singleton edge label set support”

- 4) A claim of conformance to a specific version of The Unicode® Standard and the synchronous versions of **Unicode Technical Standard #10**, **Unicode Standard Annex #15**, and **Unicode Standard Annex #31**. The claimed version of The Unicode® Standard shall not be less than “13.0.0”.

- 5) A claim of conformance to at least one of:

- Feature GA05, “Empty edge label set support”
- Feature GA06, “Singleton edge label set support”

« WG3:BER-032R3 »

- | 6) A claim conformance to Feature G001, "Repeatable-elements match mode".

24.3 Conformance to features

In addition to the data model, syntax and semantics that are mandatory for minimum conformance to this document this document defines optional features. These features are identified by Feature ID and are controlled by explicit or implicit Conformance Rules (see Subclause 5.3.7, "Feature ID and Feature Name").

An optional feature *FEAT* is defined by relaxing selected Conformance Rules, as noted at the beginning of each Conformance Rule by the phrase "without Feature *FEAT*, "name of feature", ... ". An application designates a set of GQL features that the application requires; the GQL language of the application shall observe the restrictions of all Conformance Rules except those explicitly relaxed for the required features. Conversely, conforming GQL-implementations shall identify which GQL features the GQL-implementation supports. A GQL-implementation shall process any application whose required features are a subset of the GQL-implementations supported features.

A feature *FEAT1* may imply another feature *FEAT2*. A GQL-implementation that claims to support *FEAT1* shall also support each feature *FEAT2* implied by *FEAT1*. Conversely, an application need only designate that it requires *FEAT1*, and can assume that this includes each feature *FEAT2* implied by *FEAT1*. The list of features that are implied by other features is shown in Table 8, "Implied feature relationships". Note that some features imply multiple other features.

The Syntax Rules and General Rules may define one GQL syntax in terms of another. Such transformations are presented to define the semantics of the transformed syntax and are effectively performed after checking the applicable Conformance Rules, unless otherwise noted in a Syntax Rule that defines a transformation. Transformations may use GQL syntax of one GQL feature to define another GQL feature. These transformations serve to define the behavior of the syntax, and do not have any implications for the feature syntax that is permitted or forbidden by the features so defined, except as otherwise noted in a Syntax Rule that defines a transformation. A conforming GQL-implementation need only process the untransformed syntax defined by the Conformance Rules that are applicable for the set of features that the GQL-implementation claims to support, though with the semantics implied by the transformation.

24.4 Requirements for GQL-programs

24.4.1 Introduction to requirements for GQL-programs

A conforming GQL-program shall be processed without syntax error by a conforming GQL-implementation if all of the following are satisfied:

- Every command or procedure invoked by the GQL-program is syntactically correct in accordance with this document.
- The GQL-implementation claims conformance to all the optional features to which the GQL-program claims conformance.
- The graph or graphs being processed are conforming and the conformance claims of these graphs do not include any features not included in the GQL-program's claim of conformance.

A conforming GQL application shall not use any additional features beyond the level of conformance claimed.

24.4.2 Claims of conformance for GQL-programs

A claim of conformance by a GQL-program to this document shall include all of the following:

- A claim of minimum conformance.
- Zero or more additional claims of conformance to optional features.

24.4 Requirements for GQL-programs

- A list of the implementation-defined elements and actions that are relied on for correct performance

24.5 Requirements for GQL-implementations

24.5.1 Introduction to requirements for GQL-implementations

A conforming GQL-implementation shall correctly translate and execute all GQL-programs conforming to both the GQL language and the implementation-defined features of the GQL-implementation.

A conforming GQL-implementation shall reject all GQL-programs that contain errors whose detection is required by this document.

A conforming GQL-implementation that provides optional features or that provides facilities beyond those specified in this document shall provide a GQL Flagger (See [Subclause 24.7, “GQL Flagger”](#)).

24.5.2 Claims of conformance for GQL-implementations

A claim of conformance by a GQL-implementation to this document shall include all of the following:

- 1) A claim of minimum conformance.
- 2) Zero or more additional claims of conformance to optional features.
- 3) The definition for every element and action, within the scope of the claim, that this document specifies to be implementation-defined.

24.5.3 Extensions and options

A GQL-implementation may provide implementation-defined features that are additional to those specified by this document, and may add to the list of reserved words.

NOTE 207 — If additional words are reserved, it is possible that a conforming statement cannot be processed correctly.

A GQL-implementation may provide user options to process non-conforming statements. A GQL-implementation may provide user options to process statements so as to produce a result different from that specified in this document.

It shall produce such results only when explicitly required by the user option.

It is implementation-defined ([ID081](#)) whether a GQL Flagger flags implementation-defined features.

NOTE 208 — A GQL Flagger may flag implementation-defined features using any Feature ID not defined by this document. However, there is no guarantee that some future edition of this document will not use such a Feature ID for a standard-defined feature.

NOTE 209 — The implementation-defined features flagged by a GQL Flagger may identify implementation-defined features from more than one GQL-implementation.

NOTE 210 — The allocation of a Feature ID to an implementation-defined feature may differ between GQL Flaggers.

24.6 Requirements for graphs

24.6.1 Introduction to requirements for graphs

A graph shall conform to the requirements of the data model in [Subclause 4.3.4, “Graphs”](#), as modified by any optional features to which conformance is claimed.

24.6.2 Claims of conformance for graphs

A claim of conformance by a graph to this document shall include all of the following:

- 1) A claim of minimum conformance.

- 2) Zero or more additional claims of conformance to optional features.
- 3) A list of the implementation-defined values that are relied on for correct processing.

24.7 GQL Flagger

A GQL Flagger is a facility provided by a GQL-implementation that is able to identify GQL language extensions, or other GQL processing alternatives, that may be provided by a conforming GQL-implementation (see [Subclause 24.5.3, “Extensions and options”](#)).

A GQL Flagger is intended to assist in the production of GQL language that is both portable and interoperable among different conforming GQL-implementations operating under different levels of this document.

A GQL Flagger is intended to effect a static check of GQL language. There is no requirement to detect extensions that cannot be determined until the General Rules are applied.

A GQL-implementation need only flag GQL language that is not otherwise in error as far as that GQL-implementation is concerned.

NOTE 211 — If a system is processing GQL language that contains errors, then it can be very difficult within a single statement to determine what is an error and what is an extension. As one possibility, a GQL-implementation may choose to check GQL language in two steps; first through its normal syntax analyzer and secondly through the GQL Flagger. The first step produces error messages for non-standard GQL language that the GQL-implementation cannot process or recognize. The second step processes GQL language that contains no errors as far as that GQL-implementation is concerned; it detects and flags at one time all non-standard GQL language that could be processed by that GQL-implementation. Any such two-step process should be transparent to the end user.

The GQL Flagger assists identification of conforming GQL language that can perform differently in alternative processing environments provided by a conforming GQL-implementation. It also provides a tool in identifying GQL elements that may have to be modified if GQL language is moved from a non-conforming to a conforming GQL processing environment.

24.8 Implied feature relationships

The following table lists those features that imply one or more other features.

Table 8 — Implied feature relationships

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
« WG3:BER-032R3 »			
G002	Different-edges match mode	G001	Repeatable-elements match mode
« WG3:BER-030 »			
G004	Path variables	G000	Graph pattern
G005	Path search prefix in a path pattern	G000	Graph pattern
G006	Graph Pattern KEEP clause: path mode prefix	G000	Graph pattern
G007	Graph Pattern KEEP clause: path search prefix	G000	Graph pattern
G008	Graph Pattern Where	G000	Graph pattern

24.8 Implied feature relationships

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
G031	Path Multiset Alternation: variable length path operands	G030	Path Multiset Alternation
G033	Path Pattern Union: variable length path operands	G032	Path Pattern Union
G039	Simplified Path Pattern Expression: full defaulting	G080	Simplified Path Pattern Expression: basic defaulting
G041	Non-local element pattern predicates	G051	Parenthesized Path Pattern: Non-local predicates
G043	Complete Full Edge Patterns	G042	Basic Full Edge Patterns
G045	Complete Abbreviated Edge Patterns	G040	Vertex pattern
G045	Complete Abbreviated Edge Patterns	G044	Basic Abbreviated Edge Patterns
G048	Parenthesized Path Pattern: Sub-path variable declaration	G038	Parenthesized path pattern expression
G049	Parenthesized Path Pattern: Path mode prefix	G038	Parenthesized path pattern expression
G050	Parenthesized Path Pattern: Where clause	G038	Parenthesized path pattern expression
G051	Parenthesized Path Pattern: Non-local predicates	G050	Parenthesized Path Pattern: Where clause
G061	Unbounded graph pattern quantifiers	G031	Path Multiset Alternation: variable length path operands
G081	Simplified Path Pattern Expression: full overrides	G082	Simplified Path Pattern Expression: basic overrides
G082	Simplified Path Pattern Expression: basic overrides	G080	Simplified Path Pattern Expression: basic defaulting
GA04	Unbounded node label set support	GA03	Singleton node label set support
GA07	Unbounded edge label set support	GA06	Singleton edge label set support

« WG3:BER-030 »

24.9 Syntactic transformations applied before Conformance Rules in SQL/PGQ

Table 9, “[Syntactic transformations applied before Conformance Rules in SQL/PGQ](#)”, identifies the syntactic transformations that are applied before applying Conformance Rules. Syntactic transformations defined by the following Syntax Rules are applied before application of Conformance Rules.

Table 9 — Syntactic transformations applied before Conformance Rules in SQL/PGQ

Subclause and Syntax Rule
Subclause 16.11, “ <simplified path pattern expression> ”, all Syntax Rules

Annex A (informative)

GQL Conformance Summary

The contents of this Annex summarizes all the Conformance Rules.

Most optional Features of this document are specified by Conformance Rules in Subclauses, however some are specified by implicit Conformance Rules in other text. These are summarized first.

- 1) Specifications for Feature GA00, “Graph label set support”.
 - a) Subclause 4.3.4, “Graphs”.
 - i) Without Feature GA00, “Graph label set support”, the graph label set of a conforming graph shall be empty.
- 2) Specifications for Feature GA01, “Graph property set support”.
 - a) Subclause 4.3.4, “Graphs”.
 - i) Without Feature GA01, “Graph property set support”, the graph property set of a conforming graph shall be empty.
- 3) Specifications for Feature GA10, “Undirected edge patterns”.
 - a) Subclause 4.3.4, “Graphs”.
 - i) Without Feature GA10, “Undirected edge patterns”, a conforming graph shall not contain undirected edges.
- 4) Specifications for Feature GA02, “Empty node label set support”.
 - a) Subclause 4.3.4, “Graphs”.
 - i) Without Feature GA02, “Empty node label set support”, a node in a conforming graph shall contain a non-empty node label set.
- 5) Specifications for Feature GA03, “Singleton node label set support”.
 - a) Subclause 4.3.4, “Graphs”.
 - i) Without Feature GA03, “Singleton node label set support”, a node in a conforming graph shall not contain a singleton node label set.
- 6) Specifications for Feature GA04, “Unbounded node label set support”.
 - a) Subclause 4.3.4, “Graphs”.
 - i) Without Feature GA04, “Unbounded node label set support”, a node in a conforming graph shall not contain a node label set that comprises one or more labels.
- 7) Specifications for Feature GA05, “Empty edge label set support”.
 - a) Subclause 4.3.4, “Graphs”.

- i) Without Feature GA05, “Empty edge label set support”, an edge in a conforming graph shall contain a non-empty edge label set.
- 8) Specifications for Feature GA06, “Singleton edge label set support”.
- a) Subclause 4.3.4, “Graphs”.
- i) Without Feature GA06, “Singleton edge label set support”, an edge in a conforming graph shall not contain a singleton edge label set.
- 9) Specifications for Feature GA07, “Unbounded edge label set support”.
- a) Subclause 4.3.4, “Graphs”.
- i) Without Feature GA07, “Unbounded edge label set support”, an edge in a conforming graph shall not contain an edge label set that comprises one or more labels.
- 10) Specifications for Feature GA08, “Named node types in graph types”.
- a) Subclause 4.14, “Graph types”.
- i) Without Feature GA08, “Named node types in graph types”, a conforming graph type shall contain an empty node type name dictionary.
- 11) Specifications for Feature GA09, “Named edge types in graph types”.
- a) Subclause 4.14, “Graph types”.
- i) Without Feature GA09, “Named edge types in graph types”, a conforming graph type shall contain an empty edge type name dictionary.

The remainder of this Annex recapitulates the Conformance Rules specified in Subclauses throughout this document, organized by feature name and Subclause.

- 1) Specifications for Feature G000, “Graph pattern”:
- a) Subclause 16.5, “<graph pattern>”:
- i) Without Feature G000, “Graph pattern”, conforming SQL language shall not contain a <graph pattern>.
- 2) Specifications for Feature G001, “Repeatable-elements match mode”:
- a) Subclause 16.5, “<graph pattern>”:
- i) Without Feature G001, “Repeatable-elements match mode”, conforming SQL language shall not contain a <repeatable elements match mode>.
- 3) Specifications for Feature G002, “Different-edges match mode”:
- a) Subclause 16.5, “<graph pattern>”:
- i) Without Feature G002, “Different-edges match mode”, conforming SQL language shall not contain a <different edges match mode>.
- 4) Specifications for Feature G003, “Explicit REPEATABLE ELEMENTS keyword”:
- a) Subclause 16.5, “<graph pattern>”:
- i) Without Feature G003, “Explicit REPEATABLE ELEMENTS keyword”, conforming SQL language shall not contain a <match mode> that specifies REPEATABLE ELEMENTS or REPEATABLE ELEMENT BINDINGS.
- 5) Specifications for Feature G004, “Path variables”:

IWD 39075:202y(E)
A GQL Conformance Summary

- a) Subclause 16.5, “<graph pattern>”:
 - i) Without Feature G004, “Path variables”, conforming SQL language shall not contain a <path pattern> that simply contains a <path variable declaration>.
- 6) Specifications for Feature G005, “Path search prefix in a path pattern”:
 - a) Subclause 16.5, “<graph pattern>”:
 - i) Without Feature G005, “Path search prefix in a path pattern”, conforming SQL language shall not contain a <path pattern> that simply contains a <path pattern prefix> that is a <path search prefix>.
- 7) Specifications for Feature G006, “Graph Pattern KEEP clause: path mode prefix”:
 - a) Subclause 16.5, “<graph pattern>”:
 - i) Without Feature G006, “Graph Pattern KEEP clause: path mode prefix”, conforming SQL language shall not contain a <keep clause>.
- 8) Specifications for Feature G007, “Graph Pattern KEEP clause: path search prefix”:
 - a) Subclause 16.5, “<graph pattern>”:
 - i) Without Feature G007, “Graph Pattern KEEP clause: path search prefix”, conforming SQL language shall not contain a <keep clause> that simply contains a <path search prefix>.
- 9) Specifications for Feature G008, “Graph Pattern Where”:
 - a) Subclause 16.5, “<graph pattern>”:
 - i) Without Feature G008, “Graph Pattern Where”, conforming SQL language shall not contain a <graph pattern where clause>.
- 10) Specifications for Feature G010, “Explicit WALK keyword”:
 - a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G010, “Explicit WALK keyword”, conforming SQL language shall not contain a <path mode> that specifies WALK.
- 11) Specifications for Feature G011, “Advanced Path Modes: TRAIL”:
 - a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G011, “Advanced Path Modes: TRAIL”, conforming SQL language shall not contain <path mode> that specifies TRAIL.
- 12) Specifications for Feature G012, “Advanced Path Modes: SIMPLE”:
 - a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G012, “Advanced Path Modes: SIMPLE”, conforming SQL language shall not contain <path mode> that specifies SIMPLE.
- 13) Specifications for Feature G013, “Advanced Path Modes: ACYCLIC”:
 - a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G013, “Advanced Path Modes: ACYCLIC”, conforming SQL language shall not contain <path mode> that specifies ACYCLIC.
- 14) Specifications for Feature G014, “Explicit PATH/PATHS keywords”:

- a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G014, “Explicit PATH/PATHS keywords”, conforming SQL language shall not contain a <path or paths>.
- 15) Specifications for Feature G015, “All path search: explicit ALL keyword”:
 - a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G015, “All path search: explicit ALL keyword”, conforming SQL language shall not contain an <all path search>.
- 16) Specifications for Feature G016, “Any path search”:
 - a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G016, “Any path search”, conforming SQL language shall not contain an <any path search>.
- 17) Specifications for Feature G017, “All shortest path search”:
 - a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G017, “All shortest path search”, conforming SQL language shall not contain a <shortest path search>.
- 18) Specifications for Feature G018, “Any shortest path search”:
 - a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G018, “Any shortest path search”, conforming SQL language shall not contain an <any shortest path search>.
- 19) Specifications for Feature G019, “Counted shortest path search”:
 - a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G019, “Counted shortest path search”, conforming SQL language shall not contain a <counted shortest path search>.
- 20) Specifications for Feature G020, “Counted shortest group search”:
 - a) Subclause 16.6, “<path pattern prefix>”:
 - i) Without Feature G020, “Counted shortest group search”, conforming SQL language shall not contain a <counted shortest group search>.
- 21) Specifications for Feature G030, “Path Multiset Alternation”:
 - a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G030, “Path Multiset Alternation”, conforming SQL language shall not contain a <path multiset alternation>.
- 22) Specifications for Feature G031, “Path Multiset Alternation: variable length path operands”:
 - a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G031, “Path Multiset Alternation: variable length path operands”, in conforming SQL language, an operand of a <path multiset alternation> shall be a fixed length path pattern.
- 23) Specifications for Feature G032, “Path Pattern Union”:

- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G032, “Path Pattern Union”, conforming SQL language shall not contain a <path pattern union>.
- 24) Specifications for Feature G033, “Path Pattern Union: variable length path operands”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G033, “Path Pattern Union: variable length path operands”, in conforming SQL language, an operand of a <path pattern union> shall be a fixed length path pattern.
- 25) Specifications for Feature G034, “Path concatenation”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G034, “Path concatenation”, conforming SQL language shall not contain a <path concatenation>.
- 26) Specifications for Feature G035, “Quantified Paths”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G035, “Quantified Paths”, conforming SQL language shall not contain a <quantified path primary> that does not immediately contain a <path primary> that is an <edge pattern>.
- 27) Specifications for Feature G036, “Quantified Edges”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G036, “Quantified Edges”, conforming SQL language shall not contain a <quantified path primary> that immediately contains a <path primary> that is an <edge pattern>.
- 28) Specifications for Feature G037, “Questioned Paths”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G037, “Questioned Paths”, conforming SQL language shall not contain a <questioned path primary>.
- 29) Specifications for Feature G038, “Parenthesized path pattern expression”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G038, “Parenthesized path pattern expression”, conforming SQL language shall not contain a <parenthesized path pattern expression>.
- 30) Specifications for Feature G039, “Simplified Path Pattern Expression: full defaulting”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G039, “Simplified Path Pattern Expression: full defaulting”, conforming SQL language shall not contain a <simplified path pattern expression> that is not a <simplified defaulting left>, a <simplified defaulting right>, or a <simplified defaulting any direction>.
- 31) Specifications for Feature G040, “Vertex pattern”:
- a) Subclause 16.7, “<path pattern expression>”:

- i) Without Feature G040, “Vertex pattern”, conforming SQL language shall not contain a <node pattern>.
- 32) Specifications for Feature G041, “Non-local element pattern predicates”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G041, “Non-local element pattern predicates”, in conforming SQL language, the <element pattern where clause> of an <element pattern> *EP* shall only reference the <element variable> declared in *EP*.
- 33) Specifications for Feature G042, “Basic Full Edge Patterns”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G042, “Basic Full Edge Patterns”, conforming SQL language shall not contain a <full edge any direction>, a <full edge pointing left>, or a <full edge pointing right>.
- 34) Specifications for Feature G043, “Complete Full Edge Patterns”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G043, “Complete Full Edge Patterns”, conforming SQL language shall not contain a <full edge pattern> that is not a <full edge any direction>, a <full edge pointing left>, or a <full edge pointing right>.
- 35) Specifications for Feature G044, “Basic Abbreviated Edge Patterns”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G044, “Basic Abbreviated Edge Patterns”, conforming SQL language shall not contain an <abbreviated edge pattern> that is a <minus sign>, <left arrow>, or <right arrow>.
- 36) Specifications for Feature G045, “Complete Abbreviated Edge Patterns”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G045, “Complete Abbreviated Edge Patterns”, conforming SQL language shall not contain an <abbreviated edge pattern> that is not a <minus sign>, <left arrow>, or <right arrow>.
- 37) Specifications for Feature G046, “Relaxed topological consistency: Adjacent vertex patterns”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G046, “Relaxed topological consistency: Adjacent vertex patterns”, in conforming SQL language, between any two <node pattern>s contained in a <path pattern expression> there shall be at least one <edge pattern>, <left paren>, or <right paren>.
- 38) Specifications for Feature G047, “Relaxed topological consistency: Concise edge patterns”:
- a) Subclause 16.7, “<path pattern expression>”:
 - i) Without Feature G047, “Relaxed topological consistency: Concise edge patterns”, in conforming SQL language, any <edge pattern> shall be immediately preceded and followed by a <node pattern>.
- 39) Specifications for Feature G048, “Parenthesized Path Pattern: Subpath variable declaration”:
- a) Subclause 16.7, “<path pattern expression>”:

IWD 39075:202y(E)
A GQL Conformance Summary

- i) Without Feature G048, "Parenthesized Path Pattern: Subpath variable declaration", conforming SQL language shall not contain a <parenthesized path pattern expression> that simply contains a <subpath variable declaration>.
- 40) Specifications for Feature G049, "Parenthesized Path Pattern: Path mode prefix":
- a) **Subclause 16.7, "<path pattern expression>":**
 - i) Without Feature G049, "Parenthesized Path Pattern: Path mode prefix", conforming SQL language shall not contain a <parenthesized path pattern expression> that immediately contains a <path mode prefix>.
- 41) Specifications for Feature G050, "Parenthesized Path Pattern: Where clause":
- a) **Subclause 16.7, "<path pattern expression>":**
 - i) Without Feature G050, "Parenthesized Path Pattern: Where clause", conforming SQL language shall not contain a <parenthesized path pattern where clause>.
- 42) Specifications for Feature G051, "Parenthesized Path Pattern: Non-local predicates":
- a) **Subclause 16.7, "<path pattern expression>":**
 - i) Without Feature G051, "Parenthesized Path Pattern: Non-local predicates", in conforming SQL language, a <parenthesized path pattern where clause> simply contained in a <parenthesized path pattern expression> PPPE shall not reference an <element variable> that is not declared in PPPE.
- 43) Specifications for Feature G060, "Bounded graph pattern quantifiers":
- a) **Subclause 16.10, "<graph pattern quantifier>":**
 - i) Without Feature G060, "Bounded graph pattern quantifiers", conforming SQL language shall not contain a <fixed quantifier> or a <general quantifier> that immediately contains an <upper bound>.
- 44) Specifications for Feature G061, "Unbounded graph pattern quantifiers":
- a) **Subclause 16.10, "<graph pattern quantifier>":**
 - i) Without Feature G061, "Unbounded graph pattern quantifiers", conforming SQL language shall not contain a <graph pattern quantifier> that immediately contains <asterisk>, <plus sign>, or a <general quantifier> that does not immediately contain an <upper bound>.
- 45) Specifications for Feature G070, "Label expression: Label disjunction":
- a) **Subclause 16.9, "<label expression>":**
 - i) Without Feature G070, "Label expression: Label disjunction", conforming SQL language shall not contain a <label disjunction>.
- 46) Specifications for Feature G071, "Label expression: Label conjunction":
- a) **Subclause 16.9, "<label expression>":**
 - i) Without Feature G071, "Label expression: Label conjunction", conforming SQL language shall not contain a <label conjunction>.
- 47) Specifications for Feature G072, "Label expression: Label negation":
- a) **Subclause 16.9, "<label expression>":**

- i) Without Feature G072, "Label expression: Label negation", conforming SQL language shall not contain a <label negation>.
- 48) Specifications for Feature G073, "Label expression: Individual label name":
- a) Subclause 16.9, "<label expression>":
 - i) Without Feature G073, "Label expression: Individual label name", conforming SQL language shall not contain a <label expression> that is a <label name>.
- 49) Specifications for Feature G074, "Label expression: Wildcard label":
- a) Subclause 16.9, "<label expression>":
 - i) Without Feature G074, "Label expression: Wildcard label", conforming SQL language shall not contain a <wildcard label>.
- 50) Specifications for Feature G075, "Parenthesized label expression":
- a) Subclause 16.9, "<label expression>":
 - i) Without Feature G075, "Parenthesized label expression", conforming SQL language shall not contain a <parenthesized label expression>.
- 51) Specifications for Feature G080, "Simplified Path Pattern Expression: basic defaulting":
- a) Subclause 16.11, "<simplified path pattern expression>":
 - i) Without Feature G080, "Simplified Path Pattern Expression: basic defaulting", conforming SQL language shall not contain a <simplified defaulting left>, a <simplified defaulting right>, or a <simplified defaulting any direction>.
- 52) Specifications for Feature G081, "Simplified Path Pattern Expression: full overrides":
- a) Subclause 16.11, "<simplified path pattern expression>":
 - i) Without Feature G081, "Simplified Path Pattern Expression: full overrides", conforming SQL language shall not contain a <simplified direction override> that is not a <simplified override left>, <simplified override right>, or a <simplified override any direction>.
- 53) Specifications for Feature G082, "Simplified Path Pattern Expression: basic overrides":
- a) Subclause 16.11, "<simplified path pattern expression>":
 - i) Without Feature G082, "Simplified Path Pattern Expression: basic overrides", conforming SQL language shall not contain a <simplified override left>, a <simplified override right>, or a <simplified override any direction>.
- 54) Specifications for Feature G090, "Property reference":
- a) Subclause 19.19, "<property reference>":
 - i) Without Feature G090, "Property reference", conforming SQL language shall not contain a <property reference>.
- 55) Specifications for Feature G100, "ELEMENT_ID function":
- a) Subclause 19.23, "<element_id function>":
 - i) Without Feature G100, "ELEMENT_ID function", conforming GQL language shall not contain an <element_id function>.
- 56) Specifications for Feature G110, "IS DIRECTED predicate":

IWD 39075:202y(E)
A GQL Conformance Summary

- a) **Subclause 18.7, “<directed predicate>”:**
 - i) Without Feature G110, “IS DIRECTED predicate”, conforming GQL language shall not contain a <directed predicate>.
- 57) Specifications for Feature G111, “IS LABELED predicate”:
 - a) **Subclause 18.8, “<labeled predicate>”:**
 - i) Without Feature G111, “IS LABELED predicate”, conforming GQL language shall not contain a <labeled predicate>.
- 58) Specifications for Feature G112, “IS SOURCE and IS DESTINATION predicate”:
 - a) **Subclause 18.9, “<source/destination predicate>”:**
 - i) Without Feature G112, “IS SOURCE and IS DESTINATION predicate”, conforming GQL language shall not contain a <source/destination predicate>.
- 59) Specifications for Feature G113, “ALL_DIFFERENT predicate”:
 - a) **Subclause 18.10, “<all_different predicate>”:**
 - i) Without Feature G113, “ALL_DIFFERENT predicate”, conforming GQL language shall not contain an <all_different predicate>.
- 60) Specifications for Feature G114, “SAME predicate”:
 - a) **Subclause 18.11, “<same predicate>”:**
 - i) Without Feature G114, “SAME predicate”, conforming GQL language shall not contain a <same predicate>.
- 61) Specifications for Feature GA10, “Undirected edge patterns”:
 - a) **Subclause 13.10, “<edge type definition>”:**
 - i) Without Feature GA10, “Undirected edge patterns”, conforming GQL language shall not contain an <edge type definition> that simply contains an <edge kind> that is DIRECTED, an <endpoint pair definition> that is an <endpoint pair definition any direction>, or a <full edge type pattern> that is a <full edge type pattern any direction>.
- 62) Specifications for Feature GB00, “Long identifiers”:
 - a) **Subclause 20.6, “<token> and <separator>”:**
 - i) Without Feature GB00, “Long identifiers”, in conforming GQL language, the maximum length in characters of the representative form of a <non-delimited identifier> shall be $2^7 - 1 = 127$.
 - ii) Without Feature GB00, “Long identifiers”, in conforming GQL language, the maximum length in characters of the representative form of a <delimited identifier> shall be $2^7 - 1 = 127$.
- 63) Specifications for Feature GB01, “Double minus sign comments”:
 - a) **Subclause 20.6, “<token> and <separator>”:**
 - i) Without Feature GB01, “Double minus sign comments”, conforming GQL language shall not contain a <simple comment> introduced with a <double minus sign>.
- 64) Specifications for Feature GB02, “Double solidus comments”:

- a) Subclause 20.6, “<token> and <separator>”:
 - i) Without Feature GB02, “Double solidus comments”, conforming GQL language shall not contain a <simple comment> introduced with a <double solidus>.
- 65) Specifications for Feature GD01, “Nested record types”:
 - a) Subclause 13.13, “<property type definition>”:
 - i) Without Feature GD01, “Nested record types”, conforming language shall not contain a <property type definition> that simply contains a <record type>.
 - b) Subclause 19.18, “<field>”:
 - i) Without Feature GD01, “Nested record types”, conforming language shall not contain a <field> that simply contains a <record value constructor>.
 - c) Subclause 20.4, “<field type>”:
 - i) Without Feature GD01, “Nested record types”, conforming language shall not contain a <field type> that simply contains a <record type>.

Annex B (informative)

Implementation-defined elements

This Annex references those features that are identified in the body of this document as implementation-defined.

The term *implementation-defined* is used to identify characteristics that may differ between GQL-implementations, but that shall be defined for each particular GQL-implementation.

- 1) References to: "The manner, if it so chooses, in which a GQL-implementation automatically creates a GQL-directory. (IA001)":
 - a) Subclause 4.2.5.2, "GQL-directories":
 - i) 5th paragraph
- 2) References to: "The manner, if it so chooses, in which a GQL-implementation automatically populates a GQL-schema upon its creation. (IA002)":
 - a) Subclause 4.2.5.3, "GQL-schemas":
 - i) 5th paragraph
- 3) References to: "The result of any operation other than a normalize function or a normalized predicate on an unnormalized character string. (IA003)":
 - a) Subclause 4.16.3.2.1, "Introduction to character strings":
 - i) 4th paragraph
- 4) References to: "The rules for determining the actual value of an approximate numeric type from its apparent value. (IA004)":
 - a) Subclause 4.16.3.4.2, "Characteristics of numbers":
 - i) 5th paragraph
- 5) References to: "Whether rounding or truncating occurs when least significant digits are lost on assignment. (IA005)":
 - a) Subclause 4.16.3.4.2, "Characteristics of numbers":
 - i) 6th paragraph
 - ii) 7th paragraph
 - b) Subclause 19.22, "<cast specification>":
 - i) General Rule 3)a)i)
 - ii) General Rule 4)a)i)
 - iii) General Rule 5)a)i)
 - c) Subclause 20.1, "<literal>":

- i) General Rule 3)
- 6) References to: "The choice of value selected when there is more than one approximation for an exact numeric type that conforms to the criteria. (IA006)":
- a) Subclause 4.16.3.4.2, "Characteristics of numbers":
 - i) 9th paragraph
- 7) References to: "Which numeric values other than exact numeric types also have approximations. (IA007)":
- a) Subclause 4.16.3.4.2, "Characteristics of numbers":
 - i) 10th paragraph
- 8) References to: "The choice of value selected when there is more than one approximation for an approximate numeric type that conforms to the criteria. (IA008)":
- a) Subclause 4.16.3.4.2, "Characteristics of numbers":
 - i) 11th paragraph
- 9) References to: "The result of resolving an external object url path. (IA010)":
- a) Subclause 17.10, "<external object reference>":
 - i) General Rule 2)
- 10) References to: "Whether rounding or truncating is used on division with an approximate mathematical result. (IA011)":
- a) Subclause 19.4, "<numeric value expression>":
 - i) General Rule 8)b)
- 11) References to: "The manner in which the result of the concatenation of non-normalized character strings is determined. (IA012)":
- a) Subclause 19.7, "<string value expression>":
 - i) General Rule 2)b)ii)2)B)
- 12) References to: "Whether the General Rules of Evaluation of a selective path pattern are terminated if an exception condition is raised. (IA013)":
- a) Subclause 21.4, "Evaluation of a selective <path pattern>":
 - i) General Rule 2)
- 13) References to: "The decision to raise an exception condition if the declared type of NULL cannot be determined contextually. (IA014)":
- a) Subclause 20.1, "<literal>":
 - i) Syntax Rule 48)b)
- 14) References to: "Whether to pad character strings for comparison, or not. (IA015)":
- a) Subclause 18.3, "<comparison predicate>":
 - i) General Rule 5)a)
- 15) References to: "Whether to treat byte string differing on in right-most X'00' bytes as equal, or not. (IA016)":

B Implementation-defined elements

- a) Subclause 18.3, “<comparison predicate>”:
 - i) General Rule 6)c)
- 16) References to: “Whether or not an exception condition is raised or an arbitrary value is chosen when multiple assignments to a graph element property are specified. (IA017)”:
 - a) Subclause 14.4, “<set statement>”:
 - i) General Rule 4)
- 17) References to: “The rules regarding the comparison of reference values that have different referents but are of the same base type. (IA018)”:
 - a) Subclause 4.4.3, “Reference values”:
 - i) 2nd paragraph
 - b) Subclause 18.3, “<comparison predicate>”:
 - i) General Rule 10)
- 18) References to: “The object (principal) that represents a user within a GQL-implementation. (ID001)”:
 - a) Subclause 4.2.4.1, “Principals”:
 - i) 1st paragraph
- 19) References to: “The association between a principal and its home schema and home graph. (ID002)”:
 - a) Subclause 4.2.4.1, “Principals”:
 - i) 3rd paragraph
- 20) References to: “The set of privileges identified by an authorization identifier. (ID003)”:
 - a) Subclause 4.2.4.2, “Authorization identifiers”:
 - i) 2nd paragraph
- 21) References to: “The period (GQL-Session) in which consecutive GQL-requests are executed by a GQL-client on behalf of a GQL-Agent. (ID006)”:
 - a) Subclause 4.5.1, “General description of GQL-sessions”:
 - i) 1st paragraph
- 22) References to: “Any relaxation of the assumption of the serializable transactional behavior. (ID007)”:
 - a) Subclause 4.6.1, “General description of GQL-transactions”:
 - i) 5th paragraph
- 23) References to: “The set of additionally supported property value types. (ID008)”:
 - a) Subclause 4.4.2, “Property values and supported property value types”:
 - i) 3rd paragraph
- 24) References to: “Any additional restrictions, requirements, and conditions imposed on mixed-mode transactions. (ID009)”:
 - a) Subclause 4.6.1, “General description of GQL-transactions”:
 - i) 7th paragraph

- 25) References to: "The conditions raised when the requirements on mixed-mode transactions are violated. (ID010)":
 - a) Subclause 4.6.1, "General description of GQL-transactions":
 - i) 7th paragraph
- 26) References to: "The levels of transaction isolation, their interactions, their granularity of application and the means of selecting them. (ID011)":
 - a) Subclause 4.6.3, "Transaction isolation":
 - i) 1st paragraph
- 27) References to: "Additional conditions for which a completion condition warning (01000) is raised. (ID015)":
 - a) Subclause 4.9.2, "Conditions":
 - i) 4th paragraph
- 28) References to: "The translation of condition texts. (ID016)":
 - a) Subclause 4.9.3, "GQL-status object":
 - i) 4th paragraph
- 29) References to: "The map of diagnostic information, if provided. (ID017)":
 - a) Subclause 4.9.3, "GQL-status object":
 - i) 5th list item, of the 3rd paragraph
- 30) References to: "The extent to which further GQL-status objects are chained. (ID018)":
 - a) Subclause 4.9.3, "GQL-status object":
 - i) 6th paragraph
- 31) References to: "The additional statements provided, if any. (ID020)":
 - a) Subclause 4.10.4.3, "Statements":
 - i) 3rd paragraph
- 32) References to: "The preferred name of the Boolean type. (ID021)":
 - a) Subclause 4.16.3.1, "Boolean types":
 - i) 2nd list item, of the 3rd paragraph
 - b) Subclause 20.2, "<value type>":
 - i) Syntax Rule 4)
- 33) References to: "The default collation. (ID022)":
 - a) Subclause 4.16.3.2.2, "Collations":
 - i) 3rd paragraph
 - b) Subclause 21.10, "Collation determination":
 - i) Syntax Rule 2)
 - ii) Syntax Rule 2)c)

- 34) References to: "The preferred name of the character string type. (ID023)":
- a) Subclause 4.16.3.2.1, "Introduction to character strings":
i) 2nd list item, of the 5th paragraph
 - b) Subclause 20.2, "<value type>":
i) Syntax Rule 8)
- 35) References to: "The preferred name of the fixed-length byte string type. (ID024)":
- a) Subclause 4.16.3.3, "Byte string types":
i) 2nd list item, of the 3rd paragraph
 - b) Subclause 20.2, "<value type>":
i) Syntax Rule 19)
- 36) References to: "Which numeric values have approximations obtained by rounding or truncation for a given approximate numeric type or a given decimal floating-point type. (ID025)":
- a) Subclause 4.16.3.4.2, "Characteristics of numbers":
i) 12th paragraph
- 37) References to: "Limits on operations on numbers performed according to the normal rules of arithmetic. (ID026)":
- a) Subclause 4.16.3.4.2, "Characteristics of numbers":
i) 14th paragraph
- 38) References to: "The preferred name of the variable-length byte string type. (ID027)":
- a) Subclause 4.16.3.3, "Byte string types":
i) 2nd list item, of the 3rd paragraph
 - b) Subclause 20.2, "<value type>":
i) Syntax Rule 19)
- 39) References to: "The binary precision of a signed regular integer type. (ID028)":
- a) Subclause 4.16.3.4.3, "Binary exact numeric types":
i) 7th list item, of the 1st paragraph
 - b) Subclause 20.2, "<value type>":
i) Syntax Rule 40)g)
- 40) References to: "The binary precision of a signed small integer type. (ID029)":
- a) Subclause 4.16.3.4.3, "Binary exact numeric types":
i) 8th list item, of the 1st paragraph
 - b) Subclause 20.2, "<value type>":
i) Syntax Rule 40)h)
- 41) References to: "The binary precision of a signed big integer type. (ID030)":

- a) Subclause 4.16.3.4.3, "Binary exact numeric types":
 - i) 9th list item, of the 1st paragraph
 - b) Subclause 20.2, "<value type>":
 - i) Syntax Rule 40)i)
- 42) References to: "The binary precision of a signed user-specified integer type. (ID031)":
- a) Subclause 4.16.3.4.3, "Binary exact numeric types":
 - i) 10th list item, of the 1st paragraph
 - b) Subclause 20.2, "<value type>":
 - i) Syntax Rule 40)j)
- 43) References to: "The binary precision of an unsigned user-specified integer type. (ID033)":
- a) Subclause 4.16.3.4.3, "Binary exact numeric types":
 - i) 10th list item, of the 2nd paragraph
 - b) Subclause 20.2, "<value type>":
 - i) Syntax Rule 41)j)
- 44) References to: "The decimal precision of a regular decimal exact numeric type. (ID034)":
- a) Subclause 4.16.3.4.4, "Decimal exact numeric types":
 - i) 1st list item, of the 1st paragraph
 - b) Subclause 20.2, "<value type>":
 - i) Syntax Rule 44)c)
- 45) References to: "The decimal precision of a user-specified decimal exact numeric type without a scale specification. (ID035)":
- a) Subclause 4.16.3.4.4, "Decimal exact numeric types":
 - i) 2nd list item, of the 1st paragraph
 - b) Subclause 20.2, "<value type>":
 - i) Syntax Rule 44)b)
- 46) References to: "The decimal precision of a user-specified decimal exact numeric type with a scale specification. (ID036)":
- a) Subclause 4.16.3.4.4, "Decimal exact numeric types":
 - i) 3rd list item, of the 1st paragraph
 - b) Subclause 20.2, "<value type>":
 - i) Syntax Rule 44)a)
- 47) References to: "The binary precision of a regular approximate numeric type. (ID037)":
- a) Subclause 4.16.3.4.5, "Approximate numeric types":
 - i) 6th list item, of the 1st paragraph

B Implementation-defined elements

- b) Subclause 20.2, “<value type>”:
 - i) Syntax Rule 48)f)
 - c) Subclause 21.11, “Result of value type combinations”:
 - i) General Rule 3)c)ii)1)
- 48) References to: “The binary scale of a regular approximate numeric type. (ID038)”:
- a) Subclause 4.16.3.4.5, “Approximate numeric types”:
 - i) 6th list item, of the 1st paragraph
 - b) Subclause 20.2, “<value type>”:
 - i) Syntax Rule 48)f)
- 49) References to: “The binary precision of a real approximate numeric type. (ID039)”:
- a) Subclause 4.16.3.4.5, “Approximate numeric types”:
 - i) 7th list item, of the 1st paragraph
 - b) Subclause 20.2, “<value type>”:
 - i) Syntax Rule 48)g)
- 50) References to: “The binary scale of a real approximate numeric type. (ID040)”:
- a) Subclause 4.16.3.4.5, “Approximate numeric types”:
 - i) 7th list item, of the 1st paragraph
 - b) Subclause 20.2, “<value type>”:
 - i) Syntax Rule 48)g)
- 51) References to: “The binary precision of a double approximate numeric type. (ID041)”:
- a) Subclause 4.16.3.4.5, “Approximate numeric types”:
 - i) 8th list item, of the 1st paragraph
 - b) Subclause 20.2, “<value type>”:
 - i) Syntax Rule 48)h)
- 52) References to: “The binary scale of a double approximate numeric type. (ID042)”:
- a) Subclause 4.16.3.4.5, “Approximate numeric types”:
 - i) 8th list item, of the 1st paragraph
 - b) Subclause 20.2, “<value type>”:
 - i) Syntax Rule 48)h)
- 53) References to: “The binary precision of a user-specified approximate numeric type without a scale specification. (ID043)”:
- a) Subclause 4.16.3.4.5, “Approximate numeric types”:
 - i) 9th list item, of the 1st paragraph
 - b) Subclause 20.2, “<value type>”:

- i) Syntax Rule 48)i)i)
- 54) References to: "The binary scale of a user-specified approximate numeric type without a scale specification. (ID044)":
- a) Subclause 4.16.3.4.5, "Approximate numeric types":
 - i) 9th list item, of the 1st paragraph
 - b) Subclause 20.2, "<value type>":
 - i) Syntax Rule 48)i)ii)
- 55) References to: "The binary precision of a user-specified approximate numeric type with a scale specification. (ID045)":
- a) Subclause 20.2, "<value type>":
 - i) Syntax Rule 48)i)ii)
- 56) References to: "The binary scale of a user-specified approximate numeric type with a scale specification. (ID046)":
- a) Subclause 4.16.3.4.5, "Approximate numeric types":
 - i) 10th list item, of the 1st paragraph
 - b) Subclause 20.2, "<value type>":
 - i) Syntax Rule 48)i)ii)
- 57) References to: "The treatment of language that does not conform to the Formats and Syntax Rules. (ID047)":
- a) Subclause 4.16.3.4.5, "Approximate numeric types":
 - i) 10th list item, of the 1st paragraph
 - b) Subclause 5.3.3.2, "Terms denoting rule requirements":
 - i) 1st paragraph
- 58) References to: "The default time zone identifier. (ID048)":
- a) Subclause 6.1, "<GQL-request>":
 - i) General Rule 1)a)iii)
- 59) References to: "The default session parameters. (ID049)":
- a) Subclause 6.1, "<GQL-request>":
 - i) General Rule 1)a)vi)
- 60) References to: "The Format and Syntax Rules for an implementation-defined access mode. (ID054)":
- a) Subclause 8.3, "<transaction characteristics>":
 - i) Syntax Rule 4)
- 61) References to: "Additional exception condition subclass codes for transaction rollback. (ID055)":
- a) Subclause 8.5, "<commit command>":
 - i) General Rule 2)b)

B Implementation-defined elements

- 62) References to: "The exact numeric type of array element ordinals. (ID057)":
 a) Subclause 15.6.6, "<for statement>":
 i) General Rule 3)e)iii)1)
- 63) References to: "The exact numeric type of array element indexes. (ID058)":
 a) Subclause 15.6.6, "<for statement>":
 i) General Rule 3)e)iii)2)
- 64) References to: "The declared type of the result of COUNT function. (ID059)":
 a) Subclause 16.19, "<aggregate function>":
 i) Syntax Rule 7)
- 65) References to: "The implicit null ordering. (ID060)":
 a) Subclause 16.20, "<sort specification list>":
 i) Syntax Rule 6)
- 66) References to: "The exception condition(s) to be raised on a failure to resolve an external object url path. (ID061)":
 a) Subclause 17.10, "<external object reference>":
 i) General Rule 2)
- 67) References to: "The declared type of an unsigned integer specification. (ID062)":
 a) Subclause 19.1, "<value specification>":
 i) Syntax Rule 5)
- 68) References to: "The declared type of the result of a dyadic arithmetic operator when either operand is approximate numeric. (ID063)":
 a) Subclause 19.4, "<numeric value expression>":
 i) General Rule 3)a)
- 69) References to: "The declared type of the result of a dyadic arithmetic operator when both operands are exact numeric. (ID064)":
 a) Subclause 19.4, "<numeric value expression>":
 i) General Rule 3)b)
- 70) References to: "The precision of the result of addition and subtraction. (ID065)":
 a) Subclause 19.4, "<numeric value expression>":
 i) General Rule 3)b)ii)
- 71) References to: "The precision of the result of multiplication. (ID066)":
 a) Subclause 19.4, "<numeric value expression>":
 i) General Rule 3)b)iii)
- 72) References to: "The precision and scale of the result of division. (ID067)":
 a) Subclause 19.4, "<numeric value expression>":

- i) General Rule 3)b)iv)
- 73) References to: "The declared type of the result of a length expression. (ID068)":
a) Subclause 19.6, "<numeric value function>":
 i) General Rule 1)
- 74) References to: "The declared type of the result of a trigonometric function. (ID069)":
a) Subclause 19.6, "<numeric value function>":
 i) General Rule 4)
- 75) References to: "The declared type of the result of a general logarithm function. (ID070)":
a) Subclause 19.6, "<numeric value function>":
 i) General Rule 5)
- 76) References to: "The declared type of the result of a natural logarithm. (ID071)":
a) Subclause 19.6, "<numeric value function>":
 i) General Rule 6)
- 77) References to: "The declared type of the result of an exponential function. (ID072)":
a) Subclause 19.6, "<numeric value function>":
 i) General Rule 7)
- 78) References to: "The declared type of the result of a power function. (ID073)":
a) Subclause 19.6, "<numeric value function>":
 i) General Rule 8)
- 79) References to: "The precision of an exact numeric result of a numeric value expression. (ID074)":
a) Subclause 19.6, "<numeric value function>":
 i) General Rule 9)a)
- 80) References to: "The precision of an approximate numeric result of a numeric value expression. (ID075)":
a) Subclause 19.6, "<numeric value function>":
 i) General Rule 9)b)
- 81) References to: "The declared type of an element_id function. (ID076)":
a) Subclause 19.23, "<element_id function>":
 i) Syntax Rule 2)
- 82) References to: "The maximum number of digits permitted in an unsigned integer literal. (ID078)":
a) Subclause 20.1, "<literal>":
 i) Syntax Rule 9)
- 83) References to: "The declared type of an approximate numeric literal. (ID079)":
a) Subclause 20.1, "<literal>":

i) Syntax Rule 12)

- 84) References to: "The decimal precision of a user-specified decimal exact numeric type with a scale specification. (ID080)":
- a) Subclause 4.16.3.4.4, "Decimal exact numeric types":
i) 3rd list item, of the 1st paragraph
 - b) Subclause 20.2, "<value type>":
i) Syntax Rule 44)a)
- 85) References to: "Whether a GQL Flagger flags implementation-defined features. (ID081)":
- a) Subclause 24.5.3, "Extensions and options":
i) 4th paragraph
- 86) References to: "The maximum precision of an exact numeric type. (ID082)":
- a) Subclause 20.2, "<value type>":
i) Syntax Rule 47)a)
- 87) References to: "The maximum scale of an exact numeric type. (ID083)":
- a) Subclause 20.2, "<value type>":
i) Syntax Rule 47)b)
- 88) References to: "The character (code) interpreted as newline. (ID084)":
- a) Subclause 20.6, "<token> and <separator>":
i) Syntax Rule 10)
- 89) References to: "The declared type of NULL if its declared type cannot be determined contextually. (ID085)":
- a) Subclause 20.1, "<literal>":
i) Syntax Rule 48)b)
- 90) References to: "The default merge mode. (ID086)":
- a) Subclause 4.11.9, "<match mode>":
i) 3rd paragraph
 - b) Subclause 16.5, "<graph pattern>":
i) Syntax Rule 10)
- 91) References to: "The preferred name of node types (either NODE or VERTEX). (ID087)":
- a) Subclause 13.9, "<node type definition>":
i) General Rule 2)
- 92) References to: "The preferred name of edge types (either EDGE or RELATIONSHIP). (ID088)":
- a) Subclause 13.10, "<edge type definition>":
i) General Rule 7)

- 93) References to: "The preferred name of node reference value types (either NODE or VERTEX). (ID089)":
a) Subclause 20.2, "<value type>":
i) Syntax Rule 58)
- 94) References to: "The preferred name of edge reference value types (either EDGE or RELATIONSHIP). (ID090)":
a) Subclause 20.2, "<value type>":
i) Syntax Rule 62)
- 95) References to: "The preferred name of node reference base types (either NODE REFERENCE or VERTEX REFERENCE). (ID091)":
a) Subclause 20.2, "<value type>":
i) General Rule 6)a)
ii) General Rule 7)a)
- 96) References to: "The preferred name of node base types (either NODE DATA or VERTEX DATA). (ID092)":
a) Subclause 20.2, "<value type>":
i) General Rule 6)b)
ii) General Rule 7)b)
- 97) References to: "The preferred name of edge reference base types (either EDGE REFERENCE or RELATIONSHIP REFERENCE). (ID093)":
a) Subclause 20.2, "<value type>":
i) General Rule 8)a)
- 98) References to: "The maximum cardinality of a graph label set. (IL001)":
a) Subclause 4.3.4.1, "Introduction to graphs":
i) 1st list item, of the 3rd paragraph
- 99) References to: "The maximum cardinality of a graph property set. (IL002)":
a) Subclause 4.3.4.1, "Introduction to graphs":
i) 1st list item, of the 3rd paragraph
- 100) References to: "The maximum cardinality of a node label set. (IL003)":
a) Subclause 4.3.4.1, "Introduction to graphs":
i) 1st list item, of the 3rd list item, of the 3rd paragraph
- 101) References to: "The maximum cardinality of a node property set. (IL004)":
a) Subclause 4.3.4.1, "Introduction to graphs":
i) 1st list item, of the 3rd list item, of the 3rd paragraph
- 102) References to: "The maximum cardinality of an edge label set. (IL005)":

- a) Subclause 4.3.4.1, "Introduction to graphs":
 - i) 1st list item, of the 4th list item, of the 3rd paragraph
- 103) References to: "The maximum cardinality of an edge property set. (IL006)":
- a) Subclause 4.3.4.1, "Introduction to graphs":
 - i) 1st list item, of the 4th list item, of the 3rd paragraph
- 104) References to: "The maximum cardinality of a graph type label set. (IL007)":
- a) Subclause 4.14.1, "Introduction to graph types":
 - i) 1st list item, of the 1st paragraph
- 105) References to: "The maximum cardinality of a graph type property set. (IL008)":
- a) Subclause 4.14.1, "Introduction to graph types":
 - i) 1st list item, of the 1st paragraph
- 106) References to: "The maximum cardinality of a node type label set. (IL009)":
- a) Subclause 4.14.3.3, "Node types":
 - i) 1st list item, of the 2nd paragraph
- 107) References to: "The maximum cardinality of a node type property set. (IL010)":
- a) Subclause 4.14.3.3, "Node types":
 - i) 1st list item, of the 2nd paragraph
- 108) References to: "The maximum cardinality of an edge type label set. (IL011)":
- a) Subclause 4.14.3.6, "Edge types":
 - i) 1st list item, of the 2nd paragraph
- 109) References to: "The maximum cardinality of an edge type property set. (IL012)":
- a) Subclause 4.14.3.6, "Edge types":
 - i) 1st list item, of the 2nd paragraph
- 110) References to: "The maximum length of a character string. (IL013)":
- a) Subclause 4.16.3.2.1, "Introduction to character strings":
 - i) 2nd paragraph
 - b) Subclause 19.7, "<string value expression>":
 - i) General Rule 2)b)ii)3)
 - c) Subclause 19.8, "<string value function>":
 - i) General Rule 14)d)
 - ii) General Rule 15)e)
 - d) Subclause 20.2, "<value type>":
 - i) Syntax Rule 10)

ii) **Syntax Rule 13)**

111) References to: "The maximum length of a byte string. (IL014)":

- a) **Subclause 4.16.3.3, "Byte string types":**
 - i) 2nd paragraph
- b) **Subclause 19.7, "<string value expression>":**
 - i) General Rule 3)b)
- c) **Subclause 20.2, "<value type>":**
 - i) Syntax Rule 25)
 - ii) Syntax Rule 30)

112) References to: "The maximum cardinality of a list type. (IL015)":

- a) **Subclause 19.13, "<list value expression>":**
 - i) Syntax Rule 2)b)
- b) **Subclause 19.15, "<list value constructor>":**
 - i) Syntax Rule 3)

113) References to: "The maximum precision of an exact numeric type. (IL016)":

- a) **Subclause 20.2, "<value type>":**
 - i) Syntax Rule 37)a)
- b) **Subclause 21.11, "Result of value type combinations":**
 - i) General Rule 3)c)ii)2)

114) References to: "The maximum scale of an exact numeric type. (IL017)":

- a) **Subclause 20.2, "<value type>":**
 - i) Syntax Rule 37)b)

115) References to: "The maximum value of the upper bound of a general qualifier. (IL018)":

- a) **Subclause 16.10, "<graph pattern quantifier>":**
 - i) Syntax Rule 1)

116) References to: "The maximum number of record fields. (IL019)":

- a) **Subclause 20.2, "<value type>":**
 - i) Syntax Rule 74)

117) References to: "The maximum depth of nesting of GQL-directories. (IL020)":

- a) **Subclause 4.2.5.1, "General description of the GQL-catalog":**
 - i) 4th paragraph

118) References to: "The mechanism to instruct a GQL-client to create and destroy GQL-sessions to GQL-servers, and to submit GQL-requests to them. (IW001)":

- a) **Subclause 4.2.2, "GQL-agents":**

i) 1st paragraph

- 119) References to: "The means of creating and destroying authorization identifiers, and their mapping to principals. (IW002)":
- Subclause 4.2.4.1, "Principals":
- 2nd paragraph
- 120) References to: "The mechanism and rules by which a GQL-implementation determines when the last request has been received. (IW003)":
- Subclause 4.5.1, "General description of GQL-sessions":
- 2nd paragraph
- 121) References to: "Alternative means of starting and terminating of transactions. (IW004)":
- Subclause 4.6.2, "Transaction demarcation":
- 1st paragraph
- 122) References to: "The way in which termination success or failure statuses are made available to the GQL-agent or administrator. (IW005)":
- Subclause 4.6.2, "Transaction demarcation":
- 10th paragraph
- 123) References to: "The way of determining the dictionary of request parameters of a GQL-request. (IW006)":
- Subclause 4.7.1, "General description of GQL-requests":
- 2nd list item, of the 2nd paragraph
- Subclause 4.7.2.2, "GQL-request context creation":
- 1st paragraph
- 124) References to: "How a GQL-status object is presented to a GQL-client. (IW007)":
- Subclause 4.9.1, "Introduction to diagnostics":
- 3rd paragraph
- 125) References to: "The way of determining the graph type of a graph without an explicitly specified graph type. (IW008)":
- Subclause 19.1, "<value specification>":
- Syntax Rule 3
- 126) References to: "The way in which a binding table type is inferred. (IW009)":
- Subclause 19.1, "<value specification>":
- Syntax Rule 4
- 127) References to: "The mechanism by which an external procedure is provided. (IW010)":
- Subclause 4.10.2.8, "Procedures classified by type of provisioning":
- 2nd list item, of the 1st paragraph

Annex C

(informative)

Implementation-dependent elements

This Annex references those places where this document states explicitly that the actions of a conforming GQL-implementation are implementation-dependent.

The term *implementation-dependent* is used to identify characteristics that may differ between GQL-implementations, but that are not necessarily specified for any particular GQL-implementation.

- 1) References to: "The interaction between multiple GQL-environments. (UA001)":
 a) Subclause 4.2.1, "General description of GQL-environments":
 i) 2nd paragraph
- 2) References to: "Whether or not diagnostic information pertaining to more than one condition is made available. (UA002)":
 a) Subclause 4.9.2, "Conditions":
 i) 10th paragraph
- 3) References to: "The actual order of expression evaluation. (UA003)":
 a) Subclause 5.3.3.3, "Rule evaluation order":
 i) 4th paragraph
- 4) References to: "Whether or not that exception condition is actually raised when the evaluation of an inessential part of an expression or search condition would cause an exception to be raised. (UA004)":
 a) Subclause 5.3.3.3, "Rule evaluation order":
 i) 10th paragraph
- 5) References to: "Which path bindings are retained in an any paths search if the number of candidates exceeds the required number. (UA005)":
 a) Subclause 21.4, "Evaluation of a selective <path pattern>":
 i) General Rule 13)a)ii)
- 6) References to: "Which additional path bindings are actually probed to establish whether they might also raise an exception when the GQL-implementation has terminated the evaluation of a selective path pattern. (UA006)":
 a) Subclause 21.4, "Evaluation of a selective <path pattern>":
 i) General Rule 2)
- 7) References to: "The choice of the normal form of binding table type from amongst the equivalent binding table types. (UA007)":
 a) Subclause 11.4, "<binding table type>":

- i) Syntax Rule 3)
- 8) References to: "The choice of the normal form of an endpoint pair from amongst the equivalent endpoint pairs. (UA008)":
- a) Subclause 13.10, "<edge type definition>":
- i) Syntax Rule 8)
- 9) References to: "The choice of the normal form of label set definition from amongst the equivalent label set definitions. (UA009)":
- a) Subclause 13.11, "<label set definition>":
- i) Syntax Rule 4)
- 10) References to: "The choice of the normal form of property type set definition from amongst the equivalent property type set definitions. (UA010)":
- a) Subclause 13.12, "<property type set definition>":
- i) Syntax Rule 3)
- 11) References to: "The choice of the normal form of record type from amongst the equivalent record types. (UA011)":
- a) Subclause 20.2, "<value type>":
- i) Syntax Rule 71)
- 12) References to: "The sequence of rows in an unordered binding table. (US001)":
- a) Subclause 4.3.5, "Binding tables":
- i) 8th paragraph
 - ii) 9th paragraph
- b) Subclause 16.21, "<limit clause>":
- i) General Rule 2)
- c) Subclause 16.22, "<offset clause>":
- i) General Rule 2)
- 13) References to: "The order of the groups in the result of a return statement. (US003)":
- a) Subclause 15.7.2, "<return statement>":
- i) General Rule 4)b)iv)
- 14) References to: "The order of the matches in the result of the evaluation of a graph pattern. (US004)":
- a) Subclause 16.5, "<graph pattern>":
- i) General Rule 14)
- 15) References to: "The order of path bindings that have the same number of edges. (US005)":
- a) Subclause 21.4, "Evaluation of a selective <path pattern>":
- i) General Rule 13)b)j)2)
- 16) References to: "The relative ordering of peers in a sort. (US006)":

- a) Subclause 16.20, "<sort specification list>":
 - i) General Rule 1)i)
- 17) References to: "The relative ordering of items in a sort whose comparison is Unknown. (US007)":
 - a) Subclause 16.20, "<sort specification list>":
 - i) General Rule 1)g)
- 18) References to: "The value of the result of an element id function. (UV004)":
 - a) Subclause 19.23, "<element_id function>":
 - i) General Rule 2)b)
- 19) References to: "The physical representation of an instance of a data type. (UV005)":
 - a) Subclause 4.12.1, "General introduction to data types":
 - i) 1st paragraph
- 20) References to: "The actual objects on which the Information Graph Schema is based. (UV006)":
 - a) Subclause 5.3.2, "Specification of the Information Graph Schema":
 - i) 2nd paragraph
- 21) References to: "The declared type of a site that contains an intermediate result. (UV007)":
 - a) Subclause 5.3.3.3, "Rule evaluation order":
 - i) 11th paragraph
- 22) References to: "The implicit identifiers required in path pattern evaluation. (UV008)":
 - a) Subclause 14.3, "<insert statement>":
 - i) Syntax Rule 4)c)ii)
 - b) Subclause 15.5.1, "<match statement>":
 - i) Syntax Rule 4)c)ii)
 - c) Subclause 16.7, "<path pattern expression>":
 - i) Syntax Rule 9)d)
- 23) References to: "Which arbitrary value is chosen when multiple assignments to a graph element property are specified. (UV009)":
 - a) Subclause 14.4, "<set statement>":
 - i) General Rule 4)b)
- 24) References to: "The value of an object identifier. (UV010)":
 - a) Subclause 4.3.1, "General introduction to GQL-objects":
 - i) 2nd paragraph

Annex D (informative)

GQL feature taxonomy

This Annex describes a taxonomy of features defined in this document.

Table 10, “Feature taxonomy for optional features”, contains a taxonomy of the optional features of the GQL language.

In this table, the first column contains a counter that can be used to quickly locate rows of the table; these values otherwise have no use and are not stable — that is, they are subject to change in future editions of or even Technical Corrigenda to this document without notice.

The “Feature ID” column of this table specifies the formal identification of each feature and each subfeature contained in the table.

The “Feature Name” column of this table contains a brief description of the feature or subfeature associated with the Feature ID value.

Table 10, “Feature taxonomy for optional features”, does not provide definitions of the features; the definition of those features is found in the Conformance Rules that are further summarized in [Annex A, “GQL Conformance Summary”](#).

Table 10 — Feature taxonomy for optional features

	Feature ID	Feature Name
1	G000	Graph pattern
2	G001	Repeatable-elements match mode
3	G002	Different-edges match mode
4	G003	Explicit REPEATABLE ELEMENTS keyword
5	G004	Path variables
6	G005	Path search prefix in a path pattern
7	G006	Graph Pattern KEEP clause: path mode prefix
8	G007	Graph Pattern KEEP clause: path search prefix
9	G008	Graph Pattern Where
10	G010	Explicit WALK keyword
11	G011	Advanced Path Modes: TRAIL
12	G012	Advanced Path Modes: SIMPLE

	Feature ID	Feature Name
13	G013	Advanced Path Modes: ACYCLIC
14	G014	Explicit PATH/PATHS keywords
15	G015	All path search: explicit ALL keyword
16	G016	Any path search
17	G017	All shortest path search
18	G018	Any shortest path search
19	G019	Counted shortest path search
20	G020	Counted shortest group search
21	G030	Path Multiset Alternation
22	G031	Path Multiset Alternation: variable length path operands
23	G032	Path Pattern Union
24	G033	Path Pattern Union: variable length path operands
25	G034	Path concatenation
26	G035	Quantified Paths
27	G036	Quantified Edges
28	G037	Questioned Paths
29	G038	Parenthesized path pattern expression
30	G039	Simplified Path Pattern Expression: full defaulting
31	G040	Vertex pattern
32	G041	Non-local element pattern predicates
33	G042	Basic Full Edge Patterns
34	G043	Complete Full Edge Patterns
35	G044	Basic Abbreviated Edge Patterns
36	G045	Complete Abbreviated Edge Patterns
37	G046	Relaxed topological consistency: Adjacent vertex patterns
38	G047	Relaxed topological consistency: Concise edge patterns
39	G048	Parenthesized Path Pattern: Subpath variable declaration
40	G049	Parenthesized Path Pattern: Path mode prefix

	Feature ID	Feature Name
41	G050	Parenthesized Path Pattern: Where clause
42	G051	Parenthesized Path Pattern: Non-local predicates
43	G060	Bounded graph pattern quantifiers
44	G061	Unbounded graph pattern quantifiers
45	G070	Label expression: Label disjunction
46	G071	Label expression: Label conjunction
47	G072	Label expression: Label negation
48	G073	Label expression: Individual label name
49	G074	Label expression: Wildcard label
50	G075	Parenthesized label expression
51	G080	Simplified Path Pattern Expression: basic defaulting
52	G081	Simplified Path Pattern Expression: full overrides
53	G082	Simplified Path Pattern Expression: basic overrides
54	G090	Property reference
55	G100	ELEMENT_ID function
56	G110	IS DIRECTED predicate
57	G111	IS LABELED predicate
58	G112	IS SOURCE and IS DESTINATION predicate
59	G113	ALL_DIFFERENT predicate
60	G114	SAME predicate
61	GA00	Graph label set support
62	GA01	Graph property set support
63	GA02	Empty node label set support
64	GA03	Singleton node label set support
65	GA04	Unbounded node label set support
66	GA05	Empty edge label set support
67	GA06	Singleton edge label set support
68	GA07	Unbounded edge label set support

	Feature ID	Feature Name
69	GA08	Named node types in graph types
70	GA09	Named edge types in graph types
71	GA10	Undirected edge patterns
72	GB00	Long identifiers
73	GB01	Double minus sign comments
74	GB02	Double solidus comments
75	GC01	Catalog and data statement mixing
76	GD01	Nested record types

Annex E (informative)

Maintenance and interpretation of GQL

ISO/IEC JTC 1 provides formal procedures for revision, maintenance, and interpretation of JTC 1 Standards, including creation and processing of "defect reports". Defect reports may result in technical corrigenda, amendments, interpretations, or other commentary on an existing International Standard.

A defect report may be submitted by a national standards body that is a P-member or O-member, a Liaison Organization, a member of the defect editing group for the subject document, or a working group of the committee responsible for the document. A defect identified by the user of the standard, or someone external to the committee, shall be processed via one of the official channels listed above. The submitter shall complete part 2 of the defect report form (see the Defect Report form in the Templates folder at the JTC 1 web site, as well as its attachment 1) and shall send the form to the Convenor or WG Secretariat with which the relevant defect editing group is associated.

**** Editor's Note (number 280) ****

Every time we republish SQL/Framework, we must consult the ISO Directives and/or JTC 1 Standing Document 21 to see whether the instructions have changed.

Potential new questions or new defect reports addressing the specifications of this document should be communicated to:

Secretariat, ISO/IEC JTC1/SC32
American National Standards Institute
11 West 42nd Street
New York, NY 10036
USA

Annex F (informative)

Differences with SQL

This Annex identifies the aspects of the GQL language that differ from the equivalent aspects of the SQL language.

1) Identifiers

SQL has 4 different types of identifiers:

- Regular identifiers
- Delimited identifiers
- Unicode delimited identifiers
- SQL language identifiers

GQL only has two types:

- Non-delimited identifiers
- Delimited identifiers

In SQL, “SQL language identifier” is only used to name character sets and since GQL only recognizes the Unicode character set this form of identifier is unnecessary in GQL.

In both SQL and GQL, a <regular identifier> is an <identifier start> followed by zero or more *identifier parts* (where an identifier part is either an <identifier start> or an <identifier extend>). GQL defines an additional element for specifying a non-delimited identifier that is called an <extended identifier> and that starts with <identifier extend> instead of <identifier start>.

However, the definitions of <identifier start> and <identifier extend> differ slightly between the two languages.

In SQL, <identifier start> is any character in the Unicode General Category classes “Lu”, “Ll”, “Lt”, “Lm”, “Lo”, and “Nl” (i.e., upper-case letters, lower-case letters, title-case letters, modifier letters, other letters, and letter numbers).

In GQL, <identifier start> is any character in the Unicode property ID_Start (this includes the Unicode General Category classes “Lu”, “Ll”, “Lt”, “Lm”, “Lo”, and “Nl” but also those with the Unicode property Other_ID_Start but none of which have the Unicode properties Pattern_Syntax or Pattern_White_Space).

Thus, GQL excludes U+2E2F (VERTICAL TILDE) from “Lm”, which SQL allows, but adds U+1885, U+1886, U+2118, U+212E, U+309B, and U+309C (MONGOLIAN LETTER ALI GALI BALUDA, MONGOLIAN LETTER ALI GAL, SCRIPT CAPITAL P, ESTIMATED SYMBOL, KATAKANA-HIRAGANA VOICED SOUND MARK, and KATAKANA-HIRAGANA SEMI-VOICED SOUND MARK, respectively), which SQL does not.

In SQL, <identifier extend> is U+00B7 (Middle Dot), or any character in the Unicode General Category classes “Mn”, “Mc”, “Nd”, “Pc”, and “Cf” (i.e., non-spacing marks, spacing combining marks, decimal numbers, connector punctuations, and formatting codes).

In GQL, <identifier extend> is any character in the Unicode property ID_Continue (this includes the Unicode General Category classes “Mn”, “Mc”, “Nd”, and “Pc” but also those with the Unicode property Other_ID_Continue but none of which have the Unicode properties Pattern_Syntax or Pattern_White_Space).

Thus, GQL excludes the characters in “Cf”, which SQL allows, but adds U+0387, U+1369...U+1371, and U+19DA (GREEK ANO TELEIA, ETHIOPIC DIGIT ONE...ETHIOPIC DIGIT NINE, and NEW TAI LUE THAM DIGIT ONE, respectively), which SQL does not.

In GQL, the non-delimited identifiers are case-sensitive, in SQL they are not.

In SQL non-delimited identifiers are limited to either 18 characters or 128 characters depending on the Feature supported and delimited identifiers are limited to either 18 or 128 characters depending on the Feature supported.

In GQL, the length of identifiers is limited to either 127 or 16383 characters depending on the Feature supported.

SQL distinguishes between “delimited identifiers” and “Unicode delimited identifiers” but GQL, because it is Unicode based, does not. GQL allows <escaped character>s including <unicode escape value>s in a delimited identifier whereas SQL does not. SQL, in “Unicode delimited identifiers”, allows the specification of an escape character (which in Unicode and GQL is “\”) and its <unicode escape value>s do not completely conform to the Unicode specification.

2) Binding tables in GQL are defined as collections of records whose fields are values of some data type. Records may have zero fields but are not permitted to have multiple fields with the equivalent name. Consequently, a binding table may have zero columns but is not permitted to have multiple columns with the same column name. Furthermore, two binding tables with the same collection of records that only differ in their canonical column sequence are considered equal by most operations.

3) Data types

In GQL, the term data type is used more broadly: It encompasses both value types (what SQL calls a “data type”) and object types. Furthermore, references to objects may be represented by reference values in GQL.

a) Character String

SQL has 3 different types of character strings

- CHARACTER
- CHARACTER VARYING
- CHARACTER LARGE OBJECT

All of these have a variety of ways of being specified:

- CHARACTER may also be expressed as CHAR, NATIONAL CHARACTER, NATIONAL CHAR, or NCHAR.
- CHARACTER VARYING may also be expressed as: CHAR VARYING, VARCHAR, NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING, or NCHAR VARYING.
- CHARACTER LARGE OBJECT may also be expressed as: CHAR LARGE OBJECT or CLOB, NATIONAL CHARACTER LARGE OBJECT, NCHAR LARGE OBJECT, or NCLOB, but surprisingly not NATIONAL CHAR LARGE OBJECT.

All of the types may specify a character set and a collation, except when the keyword includes NATIONAL, in which case the character set is implicit.

SQL allows support of zero-length character strings to be optional, i.e., implementation-defined.

GQL only has one character string type, specified as either STRING or as VARCHAR. This is roughly equivalent to SQL's CHARACTER VARYING.

GQL requires support for zero-length character strings and considers them to be different from the null value.

GQL supports only one character set: UNICODE.

GQL only currently supports a single implementation-defined collation. It does, however, specify two collations: UCS_BASIC and UNICODE, which are identical to the SQL collations of the same name.

**** Editor's Note (number 281) ****

Consider explicit support for additional collations other than UCS_BASIC and UNICODE (e.g., case insensitive variants, use of alternative collation tables). See [Language Opportunity GQL-012](#).

GQL does not currently support character large objects.

**** Editor's Note (number 282) ****

GQL currently only supports character strings whose length is $\leq 2^{31}-1$ (the Java limit). See [Language Opportunity GQL-171](#).

b) Byte String

SQL has 3 different types of byte strings

- BINARY
- BINARY VARYING
- BINARY LARGE OBJECT

All of these have a variety of ways of being specified:

- BINARY may also be expressed as: BINARY FIXED.
- BINARY VARYING may also be expressed as: VARBINARY.
- BINARY LARGE OBJECT may also be expressed as: BLOB.

GQL only has one byte string type but distinguishes between fixed-length byte string types specified as either BYTES or as BINARY and variable-length byte string types specified as either BYTES or as VARBINARY. This is roughly equivalent to SQL's BINARY FIXED and BINARY VARYING.

c) JSON

SQL supports a JSON type. GQL does not but provides its own method of support for nested data whose information model is a superset of JSON extending it with a richer set of containers as well as allowing all supported value types as the attributes of records (its equivalent of "JSON objects").

d) Row types

SQL support for row types shows up in multiple places including explicit ROW constructors, row subqueries and in multiple predicates. For example, the SQL <null predicate> has a special case to loop through all of the elements in a row and determine that an instance of a Row type is NULL if all of the elements in that instance are NULL. GQL does not include row types. GQL records are considered to be NOT NULL if they exist, even if all fields in a record are NULL.

« WG3:BER-094R1 »

- e) GQL does not support multisets; SQL does.
- 4) The type system of GQL is based on subtyping: Structural subtyping is used for record types, graph element content types, and the graph types they constitute and extends to reference value types for references to graphs and graph elements. Furthermore, every data type of GQL is organized in a type hierarchy that includes values, reference values, and the collections containing them, as well as objects.

- 5) Parameter specification in functions

SQL uses embedded keywords in the parameter list of *some* functions. GQL uses *only* comma separation of procedure parameters.

- 6) Terminal characters

SQL allows only <standard digit>s as <digit>s. GQL allows any Unicode character in the Unicode General Category class "Nd".

SQL uses <right bracket trigraph> and <left bracket trigraph> but, since GQL requires Unicode support, these trigraphs not needed and therefore not supported.

« WG3:BER-094R1 »

- 7) <boolean value expression>

GQL includes the Boolean operator XOR (exclusive disjunction). SQL does not include XOR.

- 8) <string value function>

SQL has various styles of regular expression substring functions but GQL does not have any. See Language Opportunity **GQL-032**.

- 9) <trim function>

In GQL, the TRIM function parameters are (<trim source> [<comma> <trim specification> [<trim character string>]) while in SQL, the TRIM function parameters are ([[<trim specification>] [<trim character>] FROM] <trim source>).

« WG3:BER-094R1 »

- 10) <list value function>

In GQL the trim function of lists is called TRIM. In SQL it is called TRIM_ARRAY.

- 11) <list value constructor>

- In GQL the leading keyword is allowed to be LIST and ARRAY. In SQL, the leading keyword is only ARRAY.
- In GQL the bracketing of <list element list> is only with brackets. In SQL, the bracketing of corresponding <array element list> is with brackets or bracket trigraphs.
- GQL does not support directly constructing a list from a tabular subquery; SQL does.

- 12) <comparison predicate>

SQL has collations that allow the specification of NO PAD or PAD SPACE. Since GQL does not (currently) support collations, every comparison uses PAD SPACE.

- 13) <labeled predicate>

In GQL, <labeled predicate> supports two forms of introducer for <label expression>: <colon> and IS [NOT] LABELED. In SQL, <labeled predicate> supports one form: IS [NOT] LABELED.

14) <as clause>

In SQL, in a <derived column>, the keyword AS is optional. In GQL, in a <select item alias> in a <select item>, the keyword AS is not optional.

15) <set quantifier>

In SQL, the <set quantifier> INTERSECT takes precedence over the <set quantifier>s UNION and EXCEPT, cf. <query expression body>. GQL does not specify the precedence between the INTERSECT, UNION, and EXCEPT operators but prohibits mixed sequences of these operators without explicit nesting, cf. <composite query expression>. In GQL, the effective precedence between these operators is always explicit in the query.

16) SQL includes two forms of comments, “Double minus sign comments” (required) and “Bracketed comments” (via advanced Feature T351). GQL supports three forms of comments, “Bracketed comments” (required), “Double minus sign comments” (via advanced Feature GB01), and “Double solidus comments” (via advanced Feature GB02).

Annex G (informative)

Graph serialization format

**** Editor's Note (number 283) ****

Further discussion is required regarding this topic; it is an editorial suggestion only. See Possible Problem [GQL-170](#).

This Annex identifies how parts of the GQL language can be used to serialize graphs.

A serialized graph specifies:

- 1) Optional graph labels that specify every graph label of the serialized graph.
- 2) Optional graph properties that specify every graph property of the serialized graph.
- 3) One optional graph type that specifies the graph type of the serialized graph.
- 4) Optional graph patterns that specify every graph element of the serialized graph.

A graph may be serialized as a character string that is expected to conform to the Format and Syntax specified in this document for a <serialized graph> of the GQL language.

```

<serialized graph> ::==
  [ PROPERTY ] GRAPH [ [ { <identifier> <period> }... ] <identifier> ]
  [ <of graph type> ]
  <left brace>
    [ <graph attribute specification> ] [ <simple graph pattern> ]
  <right brace>
« WG3:BER-036 »

<graph attribute specification> ::=
  [ <label set serialization> ]
  <left brace> [ <literal property specification list> ] <right brace>

<label set serialization> ::=
  <is or colon> <label set specification>

<literal property specification list> ::=
  <literal property specification> [ { <comma> <literal property specification> }... ]
« WG3:BER-040R3 »

<literal property specification> ::=
  <property name> <colon> <literal> [ [ <of type> ] <value type> ]

```

A <serialized graph> may be deserialized by creating a new (empty) graph of the specified graph type, setting every specified label, setting every specified property, and then inserting every specified graph pattern as if using an <insert statement>.

Bibliography

- [1] ISO/IEC 646, *Information technology — ISO 7-bit coded character set for information interchange*
- [2] ISO/IEC 6429, *Information technology — Control functions for coded character sets*
- [3] Freed, N. & Dürst, M.. *Character sets* [online]. Los Angeles, California, USA: Internet Assigned Numbers Authority, Available at <http://www.iana.org/assignments/character-sets>

Index

Index entries appearing in **boldface** indicate the page where the word, phrase, or BNF non-terminal was defined; index entries appearing in *italics* indicate a page where the BNF non-terminal was used in a Format; and index entries appearing in roman type indicate a page where the word, phrase, or BNF non-terminal was used in a heading, Function, Syntax Rule, Access Rule, General Rule, Conformance Rule, Table, or other descriptive text.

— A —

ABS • 335, 359, 415
 ACOS • 335, 339, 415
 ACYCLIC • 64, 228, 230, 234, 236, 237, 238, 417, 434, 468
 ADD • 415
 AGGREGATE • 201, 415
 ALL • 186, 210, 212, 213, 227, 234, 274, 275, 415
 ALL_DIFFERENT • 322, 415
 AND • 203, 329, 330, 343, 415
 ANY • 234, 235, 397, 415
 ARRAY • 399, 406, 415
 AS • 120, 121, 123, 126, 131, 144, 151, 210, 211, 215, 229, 269, 376, 377, 378, 382, 383, 384, 393, 415, 503
 ASC • 277, 415
 ASCENDING • 415
 ASIN • 335, 339, 415
 AT • 220, 415
 ATAN • 335, 339, 415
 AVG • 274, 415
 <abbreviated edge pattern> • 15, 56, 59, 240, **241**, 244, 249, 250, 439, 471
 <abbreviated edge type pattern> • 157, **158**, 159, 160
 <abbreviated edge type pattern any direction> • 158, 159, 160
 <abbreviated edge type pattern pointing left> • 158
 <abbreviated edge type pattern pointing right> • 158
 <absolute url path> • 142, 143, 282, **299**, 300, 304
 <absolute value expression> • 335, 338
 <accent quoted character representation> • 388, 391, 392
 active *GQL-transaction* • 41, 106, 145
 acyclic • 58
 <aggregate function> • 4, 54, 201, 211, 213, 247, **274**, 275, 334, 450
 <aggregate statement> • 55, 138, **201**, 456
 aggregating • 211
 alias name • 211
 <all path search> • 234, 235, 237, 238, 469
 <all shortest path search> • 64, **234**, 235
 <all_different predicate> • 309, 310, **322**, 474

allowing duplicates • 35
 alphabet • 431
 always causes • 57
 <ambient linear data-modifying statement> • 115, 147, **169**
 <ambient linear data-modifying statement body> • **169**, 170
 <ambient linear query statement> • 115, 185, **189**
 ambient statement • 54
 <ampersand> • 59, 175, 252, 257, 424, **425**
 annotated path • 431
 anonymous edge symbol • 430
 anonymous node symbol • 430
 any object type • 68
 <any path search> • 64, **234**, 235, 238, 443, 469
 any property value type • 69
 <any shortest path search> • **234**, 235, 238, 469
 any type • 68
 any value type • 68
 apparent value • 84
 applied • 432
 approximate numbers • 67
 <approximate numeric literal> • 84, 379, 380, 389, **390**, 391, 395
 <approximate numeric type> • 83, **398**, 402, 404, 405, 408
 approximate numeric types • 67
 <arc type any direction> • 157
 <arc type filler> • 157
 <arc type pointing left> • 157
 <arc type pointing right> • 157
 array • 76
 <as graph type> • 131, 151
 <as or equals> • 120, 123, 126, 131
 assignment • 88
 <asterisk> • 210, 254, 255, 274, 332, 333, 357, 358, 424, **425**, 472
 <at schema clause> • 43, 115, **220**
 at the same depth of graph pattern matching • 226
 atomic • 66
 authorization identifier • 28

— B —

BIG • 67, 85, 86, 398, 402, 403, 404, 415
 BIGINT • 67, 85, 398, 402, 403, 415
 BINARY • 66, 67, 82, 397, 401, 407, 415, 501
 BINDING • 123, 133, 417
 BINDINGS • 225, 233, 417, 467
 BNF non-terminal symbol • 89
 BOOL • 38, 66, 80, 397, 400, 415
 BOOLEAN • 38, 66, 80, 397, 400, 407, 415
 BOTH • 345, 350, 351, 415
 BY • 52, 55, 201, 206, 212, 271, 273, 415
 BYTES • 67, 82, 397, 401, 415, 501
 BYTE_LENGTH • 335, 415
 Boolean type • 80, 400
 Boolean types • 66
 Booleans • 66
 base type • 65
 base type name prefix • 65
 be included in • 94
 begin subpath symbol • 430
 <binary digit> • 389, 391, 424
 <binary exact numeric type> • 398, 402
 <binary set function> • 274, 275, 276
 <binary set function type> • 274
 binding • 231, 442
 <binding table initializer> • 123, 124, 125
 <binding table name> • 290, 291, 412, 413
 <binding table parameter definition> • 118, 123, 124, 125
 <binding table parent specification> • 290, 291
 <binding table reference> • 123, 125, 290
 <binding table resolution expression> • 290
 <binding table type> • 76, 123, 124, 129, 133, 491
 <binding table variable> • 123, 124
 <binding table variable declaration> • 118, 123, 124
 <binding table variable definition> • 118, 123, 124, 125
 <binding variable> • 8, 55, 56, 175, 176, 177, 179, 180, 211,
 212, 217, 221, 250, 270, 271, 334, 335, 336, 338
 <binding variable declaration> • 118
 <binding variable definition> • 53, 55, 56, 118, 119
 <binding variable definition block> • 53, 56, 115, 116, 117,
 118, 119, 199
 <binding variable name> • 8, 120, 123, 126, 221, 413
 binds • 431
 <boolean factor> • 192, 329
 <boolean literal> • 387, 391, 395
 <boolean predicand> • 329, 330
 <boolean primary> • 329, 330
 <boolean term> • 329
 <boolean test> • 329, 330
 <boolean type> • 80, 397, 400, 407
 <boolean value expression> • 80, 192, 308, 327, 329, 330,
 331, 502
 bounded quantifier • 255
 bracket index • 430

<bracket right arrow> • 157, 240, 418
 bracket symbol binding • 431
 bracket symbols • 430
 <bracket tilde right arrow> • 240, 418
 <bracketed comment> • 420, 421
 <bracketed comment contents> • 420, 421
 <bracketed comment introducer> • 420
 <bracketed comment terminator> • 420, 421
 <byte length expression> • 335
 byte string • 82
 <byte string concatenation> • 342, 343, 344
 <byte string factor> • 342, 343
 <byte string function> • 345, 348, 350
 <byte string literal> • 387, 389, 392, 393, 395, 415, 421
 <byte string primary> • 342
 <byte string trim function> • 345, 346, 347, 348, 350, 351
 <byte string trim source> • 346, 347, 348, 351
 <byte string type> • 82, 397, 401, 407
 byte string types • 67
 <byte string value expression> • 342, 343, 346, 347, 348,
 350
 byte strings • 67
 <byte substring function> • 345, 346, 347, 348, 350, 351

— C —

CALL • 136, 199, 416
 CASE • 373, 374, 416
 CAST • 58, 376, 377, 378, 382, 383, 384, 393, 416
 CATALOG • 51, 416
 CEIL • 336, 416
 CEILING • 336, 416
 CHARACTER • 416
 CHARACTER_LENGTH • 335, 382, 416
 CLASS_ORIGIN • 417
 CLEAR • 104, 416
 CLONE • 416
 CLOSE • 105, 416
 COALESCE • 373, 374, 416
 COLLECT • 274, 416
 COMMAND_FUNCTION • 417
 COMMAND_FUNCTION_CODE • 417
 COMMIT • 42, 110, 416
 CONDITION_NUMBER • 417
 CONNECTING • 158, 417
 CONSTANT • 416
 CONSTRAINT • 416
 CONSTRUCT • 416
 COPY • 131, 144, 416
 COS • 335, 339, 416
 COSH • 335, 339, 416
 COT • 335, 339, 416
 COUNT • 274, 275, 416

CREATE • 142, 144, 145, 151, 416
 CSMAXL • 400
 CURRENT_DATE • 353, 354, 416
 CURRENT_GRAPH • 324, 325, 416
 CURRENT_PROPERTY_GRAPH • 324, 325, 416
 CURRENT_ROLE • 416
 CURRENT_SCHEMA • 299, 324, 325, 416
 CURRENT_TIME • 353, 354, 416
 CURRENT_TIMESTAMP • 353, 354, 416
 CURRENT_USER • 324, 325, 416
 Cartesian product • 36
 calendar date • 24
 <call catalog-modifying procedure statement> • 138, 168
 <call data-modifying procedure statement> • 7, 138, 183
 <call procedure statement> • 52, 136, 168, 183, 193, 456
 <call query statement> • 28, 138, 193, 456
 canonical column sequence • 35
 <case abbreviation> • 373, 374
 <case expression> • 334, 373, 374, 375, 452
 <case operand> • 373, 374
 <case specification> • 373, 374, 375
 <case-insensitive non-reserved word> • 417, 422
 <case-insensitive reserved word> • 415, 422
 case-normal form • 422
 <cast operand> • 376, 377
 <cast specification> • 58, 59, 81, 334, 376, 377, 378
 <cast target> • 376
catalog and data statement mixing not supported • 112, 152, 326
 <catalog binding table parent and name> • 290
 <catalog binding table reference> • 123, 125, 290
 <catalog function parent and name> • 296
 <catalog function reference> • 296
 <catalog graph parent and name> • 27, 144, 149, 284
 <catalog graph reference> • 50, 120, 122, 284
 <catalog graph type parent and name> • 27, 151, 166, 287
 <catalog graph type reference> • 151, 287
 catalog namespace • 55
 catalog object • 30
 catalog object descriptor • 30
 <catalog object reference> • 17, 55, 126, 128, 299, 300
 <catalog procedure parent and name> • 293
 <catalog procedure reference> • 293
 <catalog schema parent and name> • 142, 143, 282
 <catalog url path> • 299, 300, 305
 <catalog-modifying procedure specification> • 111, 112, 168
 catalog-modifying side effects • 57
 <catalog-modifying statement> • 54, 111, 112, 115, 135
 <ceiling function> • 335, 336, 338, 341
 <char length expression> • 335
 <character representation> • 388
 <character string concatenation> • 342, 343
 <character string factor> • 342, 343
 <character string function> • 345, 348
 <character string literal> • 387, 391, 392, 393, 395, 418, 421, 422
 <character string primary> • 342, 343
 <character string type> • 66, 81, 397, 400, 407
 character string types • 67
 <character string value expression> • 335, 342, 343, 345, 347, 348, 349
 character strings • 67
 child execution context • 45
 <circumflex> • 425
 closed record type • 406
 collation • 24
 <collection value constructor> • 334
 <colon> • 120, 123, 126, 151, 175, 240, 319, 366, 370, 424, 425, 503, 504
 <comma> • 108, 153, 164, 175, 179, 181, 199, 201, 210, 215, 223, 225, 240, 250, 254, 266, 269, 271, 274, 277, 322, 323, 335, 336, 345, 346, 363, 365, 366, 368, 373, 397, 398, 399, 424, 425, 502, 504
 command • 52
 <comment> • 420
 <commit command> • 27, 107, 110, 456, 483
 <common logarithm> • 335, 336, 337
 <common value expression> • 327
 <comp op> • 278, 311, 312, 313
 <compact value variable definition> • 118, 201
 <compact variable declaration> • 118, 223
 <compact variable definition> • 118, 199, 223
 <compact variable definition list> • 199
 <comparison predicate> • 45, 192, 278, 309, 310, 311, 449, 450, 502
 <comparison predicate part 2> • 311, 373
 completion condition • 48
 <composite query expression> • 61, 184, 185, 188, 503
 <composite query statement> • 135, 184
 compressed binding • 431
 compressed path binding • 432
 <concatenation operator> • 342, 361, 418
 conditions • 49
 connection exception • 27
 <connector any direction> • 158
 <connector pointing right> • 158
 consistent • 433
 constitute • 92
 constructed • 66
 contain • 92
 contained in • 92
 containing • 92
 <copy graph expression> • 144
 <copy graph type expression> • 131, 132
 <counted shortest group search> • 64, 234, 235, 236, 238, 443, 469

<counted shortest path search> • 64, **234**, 235, 238, 443, 469
 <create graph statement> • 15, 138, **144**, 456
 <create graph type statement> • 138, 145, **151**, 152, 154, 456
 <create schema statement> • 138, **142**
 current X • 44
 current authorization identifier • 39
 current execution context • 44
 current execution outcome • 45
 current execution result • 45
 current execution stack • 43
 current home graph • 28
 current home schema • 28
 current principal • 39
 current request context • 39, 43
 current request outcome • 43
 current request parameters • 43
 current session context • 39
 current session graph • 39
 current session parameter flags • 39
 current session parameters • 39
 current session schema • 39
 current termination flag • 40
 current time zone identifier • 39
 current transaction • 39
 current working graph of A • 44
 current working record • 44
 current working schema of A • 44
 current working table • 44
 current x • 39, 43, 44

— D —

DATA • 65, 66, 67, 80, 81, 82, 83, 133, 153, 154, 156, 162, 400, 401, 406, 407, 408, 409, **416**
 DATE • 87, **353**, 354, 382, 383, 384, 390, 393, 399, 406, **416**
 DATETIME • 87, **353**, 354, 384, 390, 393, 399, 406, 416, 453
 DEC • 67, 86, 398, 402, 404, **416**
 DECIMAL • 66, 67, 83, 86, 398, 401, 402, 404, 407, **416**
 DEFAULT • **416**
 DEGREES • 335, 339, **416**
 DELETE • **181**, **416**
 DESC • 277, 278, **416**
 DESCENDING • **416**
 DESTINATION • 320, 321, **417**
 DETACH • **181**, 182, **416**
 DIFFERENT • 65, 225, **417**
 DIRECTED • 158, 159, 160, 162, 318, 417, **474**
 DIRECTORIES • **416**
 DIRECTORY • **416**
 DISTINCT • 12, 185, 186, 213, 274, 275, **416**
 DO • **171**, **181**, **416**
 DOUBLE • 67, 87, 399, 404, 405, **416**

DROP • 143, 146, 149, 152, 166, **416**
 DURATION • 88, 359, 382, 384, 390, 393, 394, 399, 406, 416, 453
data exception • 102, 177, 195, 204, 312, 326, 332, 333, 338, 339, 340, 341, 343, 344, 348, 349, 350, 351, 352, 354, 355, 357, 358, 359, 361, 363, 366, 377, 378, 379, 380, 381, 382, 383, 384, 385, 442, 452, 453
 data object • 31
 data type • 65
 <data-modifying procedure specification> • **111**, 112, 183
 data-modifying side effects • 57
 <data-modifying statement> • 54, 112, **135**
 data-populating side effect • 57
 data-reading statement • 54
 data-transforming statement • 54
 <date function> • **353**, 354
 <date function parameters> • **353**, 354, 355
 <date literal> • **390**, 393, 395
 <date string> • 353, 355, **390**, 393, 395, 418
 <datetime factor> • **352**
 <datetime function> • **353**, 354
 <datetime function parameters> • **353**, 355
 <datetime literal> • **390**, 393, 395
 <datetime primary> • **352**
 <datetime string> • 353, 355, **390**, 393, 395, 418
 <datetime term> • **352**, 357, 358
 <datetime value expression> • 25, 327, **352**, 357, 358
 <datetime value function> • 55, 352, **353**, 355
 <decimal exact numeric type> • **398**, 402, 404
 decimal numbers • 67
 decimal types • 67
 declared • 245
 declared names • 66
 degree of exposure • 228, 246
 degree of reference • 306
 <delete item> • **181**, 182
 <delete item list> • **181**
 <delete statement> • 19, 138, **181**, 456
 <delimited identifier> • 13, 413, 418, **420**, 421, 422, 474
 <delimiter token> • 243, 415, **418**, 421
dependent object error • 182
 <dependent value expression> • 217, 274, **275**
 descriptor • 94
 <destination node type name> • **158**, 159, 161
 <destination node type reference> • 157, **158**, 160
 <destination predicate part 2> • **320**, 373, 374
 <different edges match mode> • 58, 63, 64, **225**, 227, 228, 230, 231, 233, 467
 different-edges-matched • 230
 <digit> • 379, 380, 389, 391, 395, **424**, 502
 directed pointing left • 58
 directed pointing right • 58
 <directed predicate> • 309, 310, **318**, 474

<directed predicate part 2> • 318, 373, 374
 directionality constraint • 439
 directly contained in • 92
 directly containing • 92
 directly contains • 92
 distinct array • 77
 distinct list • 77
division by zero • 333, 339
 <do statement> • 18, 138, 171, 181, 456
 <dollar sign> • 412, 424, 425
 double approximate numeric type • 87, 405
 <double colon> • 397, 418
 <double minus sign> • 418, 420, 422, 474
 <double period> • 299, 300, 418
 <double quote> • 388, 392, 424, 425
 <double quoted character representation> • 388, 391, 392
 <double quoted character sequence> • 387, 388, 391, 420, 421
 <double solidus> • 420, 423, 475
 <doubled grave accent> • 388
 <drop graph statement> • 27, 138, 146, 149, 456
 <drop graph type statement> • 27, 138, 149, 152, 166, 456
 <drop schema statement> • 59, 138, 143
 duplicate-free • 35
 duration • 24
 <duration absolute value function> • 359
 <duration factor> • 357, 358
 <duration function> • 359
 <duration function parameters> • 359
 <duration literal> • 387, 390, 393, 394, 395
 <duration primary> • 357
 <duration string> • 359, 390, 394, 418
 <duration term> • 352, 357, 358
 <duration term 1> • 357
 <duration term 2> • 357, 358
 <duration value expression> • 25, 327, 352, 357, 358, 359
 <duration value expression 1> • 357, 358
 <duration value function> • 55, 357, 359
 dynamic site • 32

— E —

EDGE • 7, 162, 406, 409, 417, 420
 EDGES • 65, 417, 420
 ELEMENT • 225, 233, 417, 467
 ELEMENTS • 65, 225, 233, 417, 467
 ELEMENT_ID • 386, 416
 ELSE • 373, 374, 375, 416
 EMPTY_BINDING_TABLE • 324, 325, 416
 EMPTY_GRAPH • 144, 324, 325, 416
 EMPTY_PROPERTY_GRAPH • 324, 325, 416
 EMPTY_TABLE • 324, 325, 416
 END • 185, 208, 209, 373, 374, 416

ENDS • 416
 EXCEPT • 185, 186, 416, 503
 EXISTING • 416
 EXISTS • 103, 120, 121, 123, 124, 126, 127, 142, 143, 144, 145, 149, 151, 152, 166, 314, 416
 EXP • 336, 340, 341, 416
 edge • 253
 edge base type • 73
 <edge bindings or edges> • 225
 <edge kind> • 157, 158, 159, 162, 474
 <edge pattern> • 59, 60, 62, 228, 239, 240, 242, 243, 244, 245, 248, 249, 250, 258, 259, 260, 437, 438, 439, 470, 471
 <edge reference> • 320
 <edge reference value expression> • 327, 328
 <edge reference value type> • 67, 161, 399, 400, 406, 409
 edge symbol binding • 431
 <edge synonym> • 157, 225, 399, 420
 edge type • 73
 <edge type definition> • 74, 153, 154, 157, 159, 161, 162, 399, 406, 474, 492
 <edge type filler> • 157, 159, 160
 <edge type label set definition> • 157, 159, 160, 161
 <edge type name> • 153, 157, 159, 160, 161, 162
 <edge type pattern> • 157
 <edge type phrase> • 157, 159
 <edge type property type set definition> • 157, 159, 162
 edge variable • 245
 <edges synonym> • 225, 420
 effectively • 93
 <element bindings or elements> • 225
 <element pattern> • 59, 60, 61, 62, 239, 243, 244, 245, 246, 248, 250, 436, 437, 438, 439, 471
 <element pattern filler> • 59, 173, 191, 239, 240, 243
 <element pattern predicate> • 172, 191, 239, 240, 243, 250
 <element pattern where clause> • 59, 61, 62, 240, 243, 245, 246, 248, 250, 306, 431, 471
 <element property specification> • 172, 191, 240
 <element reference> • 306, 307, 318, 319, 320, 322, 323, 373, 374, 386, 431, 445, 446
 <element type definition> • 153
 <element type definition list> • 153
 <element type name> • 155, 157, 158, 159, 412
 <element variable> • 231, 232, 239, 242, 243, 245, 248, 249, 250, 306, 412, 413, 442, 445, 471, 472
 <element variable declaration> • 59, 61, 173, 191, 229, 239, 243, 245, 246, 250
 element variables • 60
 <element_id function> • 334, 386, 473, 493
 elementary binding • 431
 elementary binding of LET • 431
 <else clause> • 373, 374, 375
 empty • 243
 empty binding table • 35
empty binding table returned • 195

<empty grouping set> • 271
 end bracket symbol binding • 431
 end subpath symbol • 430
 <end transaction command> • 100, 107
 endNode • 415
 <endpoint definition> • 157, 158
 <endpoint pair definition> • 158, 159, 160, 162, 474
 <endpoint pair definition any direction> • 158, 159, 162, 474
 <endpoint pair definition pointing left> • 158
 <endpoint pair definition pointing right> • 158
 equality operation • 449
 <equals operator> • 118, 120, 175, 225, 241, 311, 312, 313, 329, 424, 425, 449, 450
 <escaped backspace> • 388, 389, 392
 <escaped carriage return> • 388, 389, 392
 <escaped character> • 388, 391, 392, 500
 <escaped double quote> • 388, 392
 <escaped form feed> • 388, 389, 392
 <escaped grave accent> • 388, 392
 <escaped newline> • 388, 389, 392
 <escaped quote> • 388, 392
 <escaped reverse solidus> • 388, 392
 <escaped tab> • 388, 392
 exact numbers • 67
 <exact numeric literal> • 84, 378, 380, 389, 390, 391, 395
 <exact numeric type> • 83, 398, 402, 404, 407
 exact numeric types • 67
 exception condition • 48
 <exclamation mark> • 59, 252, 257, 424, 425
 executing GQL-request • 38
 execution context • 44
 execution outcome • 46
 execution stack • 44
 <exists predicate> • 309, 310, 314, 315
 <exponent> • 84, 390, 395
 <exponential function> • 335, 336, 338, 340
 expression • 211
 <extended identifier> • 413, 415, 499
 exterior variable • 237
 <external object reference> • 282, 284, 285, 287, 290, 291, 293, 296, 305, 484
 <external object url> • 305
 extracted path • 432

— F —

FALSE • 329, 380, 381, 387, 395, 416
 FILTER • 172, 181, 198, 199, 204, 416
 FINAL • 417
 FIRST • 277, 278, 417
 FLOAT • 66, 67, 83, 87, 399, 401, 405, 408, 416
 FLOAT128 • 67, 86, 87, 398, 405, 416
 FLOAT16 • 67, 86, 398, 405, 416
 FLOAT256 • 67, 87, 398, 405, 416
 FLOAT32 • 67, 86, 398, 405, 416
 FLOAT64 • 67, 86, 398, 405, 416
 FLOOR • 336, 416
 FOR • 203, 204, 416
 FROM • 215, 416
 FUNCTION • 51, 296, 416
 FUNCTIONS • 416
 <factor> • 332, 357, 358
 failed outcome • 47
 <field> • 368, 369, 370, 394, 475
 <field list> • 368, 370, 394
 <field name> • 370, 411, 412, 413
 <field specification> • 368
 <field type> • 133, 399, 407, 409, 411, 475
 <field type list> • 399, 407, 411
 <field type specification> • 133, 399, 406, 407
 field value • 370
 field value expression • 370
 <filter statement> • 138, 172, 198, 456
 <fixed length> • 397, 400, 401
 fixed length path pattern • 241
 <fixed quantifier> • 246, 254, 255, 472
 floating point numbers • 67
 <floor function> • 335, 336, 338, 341
 <focused linear data-modifying statement> • 115, 169, 218
 <focused linear data-modifying statement body> • 169, 170, 218
 <focused linear query and primitive result statement part> • 189, 218
 <focused linear query statement> • 115, 185, 189, 218
 <focused linear query statement part> • 189, 218
 <focused nested data-modifying procedure specification> • 169, 218
 <focused nested query specification> • 189, 218
 <focused primitive result statement> • 189, 218
 focused statement • 54
 <fold> • 345, 348, 349, 350
 <for item> • 203, 204
 <for item alias> • 203, 204
 <for item list> • 203
 <for ordinality or index> • 203, 205
 <for statement> • 30, 138, 203, 204, 456
 <formal parameter declaration> • 223
 <formal parameter declaration list> • 223
 <formal parameter definition> • 223
 <formal parameter definition list> • 223
 <formal parameter list> • 223
 <full edge any direction> • 56, 240, 248, 440, 471
 <full edge left or right> • 240, 439
 <full edge left or undirected> • 240, 439
 <full edge pattern> • 59, 61, 240, 244, 248, 250, 439, 471
 <full edge pointing left> • 240, 248, 250, 439, 471

<full edge pointing right> • 240, 248, 250, 439, 471
 <full edge type pattern> • 157, 160, 162, 474
 <full edge type pattern any direction> • 157, 162, 474
 <full edge type pattern pointing left> • 157, 160
 <full edge type pattern pointing right> • 157, 160
 <full edge undirected> • 240, 250, 439
 <full edge undirected or right> • 240, 439
 <function name> • 296, 297, 412, 413
 <function parent specification> • 296, 297
 <function reference> • 296
 <function resolution expression> • 296

— G —

Feature G000, “Graph pattern” • 233, 463, 467
 Feature G001, “Repeatable-elements match mode” • 233, 460, 461, 463, 467
 Feature G002, “Different-edges match mode” • 233, 463, 467
 Feature G003, “Explicit REPEATABLE ELEMENTS keyword” • 233, 467
 Feature G004, “Path variables” • 233, 463, 467, 468
 Feature G005, “Path search prefix in a path pattern” • 233, 463, 468
 Feature G006, “Graph Pattern KEEP clause: path mode prefix” • 233, 463, 468
 Feature G007, “Graph Pattern KEEP clause: path search prefix” • 233, 463, 468
 Feature G008, “Graph Pattern Where” • 233, 460, 463, 468
 Feature G010, “Explicit WALK keyword” • 238, 468
 Feature G011, “Advanced Path Modes: TRAIL” • 238, 468
 Feature G012, “Advanced Path Modes: SIMPLE” • 238, 468
 Feature G013, “Advanced Path Modes: ACYCLIC” • 238, 468
 Feature G014, “Explicit PATH/PATHS keywords” • 238, 468, 469
 Feature G015, “All path search: explicit ALL keyword” • 238, 469
 Feature G016, “Any path search” • 238, 469
 Feature G017, “All shortest path search” • 238, 469
 Feature G018, “Any shortest path search” • 238, 469
 Feature G019, “Counted shortest path search” • 238, 469
 Feature G020, “Counted shortest group search” • 238, 469
 Feature G030, “Path Multiset Alternation” • 248, 464, 469
 Feature G031, “Path Multiset Alternation: variable length path operands” • 248, 464, 469
 Feature G032, “Path Pattern Union” • 248, 464, 469, 470
 Feature G033, “Path Pattern Union: variable length path operands” • 248, 464, 470
 Feature G034, “Path concatenation” • 248, 460, 470
 Feature G035, “Quantified Paths” • 248, 470
 Feature G036, “Quantified Edges” • 248, 470
 Feature G037, “Questioned Paths” • 248, 470
 Feature G038, “Parenthesized path pattern expression” • 248, 464, 470
 Feature G039, “Simplified Path Pattern Expression: full defaulting” • 248, 464, 470

Feature G040, “Vertex pattern” • 248, 460, 464, 470, 471
 Feature G041, “Non-local element pattern predicates” • 248, 464, 471
 Feature G042, “Basic Full Edge Patterns” • 248, 460, 464, 471
 Feature G043, “Complete Full Edge Patterns” • 248, 464, 471
 Feature G044, “Basic Abbreviated Edge Patterns” • 249, 464, 471
 Feature G045, “Complete Abbreviated Edge Patterns” • 249, 464, 471
 Feature G046, “Relaxed topological consistency: Adjacent vertex patterns” • 249, 471
 Feature G047, “Relaxed topological consistency: Concise edge patterns” • 249, 471
 Feature G048, “Parenthesized Path Pattern: Subpath variable declaration” • 249, 464, 471, 472
 Feature G049, “Parenthesized Path Pattern: Path mode prefix” • 249, 464, 472
 Feature G050, “Parenthesized Path Pattern: Where clause” • 249, 464, 472
 Feature G051, “Parenthesized Path Pattern: Non-local predicates” • 249, 464, 472
 Feature G060, “Bounded graph pattern quantifiers” • 255, 472
 Feature G061, “Unbounded graph pattern quantifiers” • 255, 464, 472
 Feature G070, “Label expression: Label disjunction” • 253, 460, 472
 Feature G071, “Label expression: Label conjunction” • 253, 472
 Feature G072, “Label expression: Label negation” • 253, 472, 473
 Feature G073, “Label expression: Individual label name” • 253, 460, 473
 Feature G074, “Label expression: Wildcard label” • 253, 473
 Feature G075, “Parenthesized label expression” • 253, 473
 Feature G080, “Simplified Path Pattern Expression: basic defaulting” • 260, 464, 473
 Feature G081, “Simplified Path Pattern Expression: full overrides” • 260, 464, 473
 Feature G082, “Simplified Path Pattern Expression: basic overrides” • 261, 464, 473
 Feature G090, “Property reference” • 371, 460, 473
 Feature G100, “ELEMENT_ID function” • 386, 473
 Feature G110, “IS DIRECTED predicate” • 318, 473, 474
 Feature G111, “IS LABELED predicate” • 319, 474
 Feature G112, “IS SOURCE and IS DESTINATION predicate” • 321, 474
 Feature G113, “ALL_DIFFERENT predicate” • 322, 474
 Feature G114, “SAME predicate” • 323, 474
 Feature GA00, “Graph label set support” • 33, 70, 466
 Feature GA01, “Graph property set support” • 33, 70, 466
 Feature GA02, “Empty node label set support” • 33, 72, 460, 466
 Feature GA03, “Singleton node label set support” • 33, 72, 460, 464, 466

Feature GA04, "Unbounded node label set support" • 33, 72, 464, 466
 Feature GA05, "Empty edge label set support" • 34, 73, 460, 466, 467
 Feature GA06, "Singleton edge label set support" • 34, 73, 460, 464, 467
 Feature GA07, "Unbounded edge label set support" • 34, 73, 464, 467
 Feature GA08, "Named node types in graph types" • 70, 467
 Feature GA09, "Named edge types in graph types" • 70, 467
 Feature GA10, "Undirected edge patterns" • 34, 56, 74, 162, 240, 466, 474
 Feature GB00, "Long identifiers" • 422, 474
 Feature GB01, "Double minus sign comments" • 422, 474, 503
 Feature GB02, "Double solidus comments" • 423, 474, 475, 503
 Feature GC01, "Catalog and data statement mixing" • 40, 112
 Feature GD01, "Nested record types" • 165, 370, 411, 475
 <GQL language character> • 424, 426
 <GQL special character> • 418, 424
 <GQL terminal character> • 10, 424
 GQL-agent • 26
 GQL-catalog • 28
 GQL-client • 27
 GQL-data • 31
 GQL-directory • 29
 GQL-directory descriptor • 29
 GQL-environment • 25
 GQL-implementation • 26
 GQL-object • 31
 <GQL-program> • 97, 100
 <GQL-request> • 43, 45, 48, 63, 97, 483
 GQL-request context • 42
 GQL-schema • 30
 GQL-server • 27
 GQL-session • 38
 GQL-status object • 49
 GQL-transaction • 40
 GQLSTATUS • 47, 48, 49, 416, 455, 457, 459
 GRANT • 416
 GRAPH • 120, 121, 131, 132, 144, 145, 146, 147, 149, 151, 152, 153, 154, 166, 284, 287, 417, 504
 GRAPHS • 417
 GROUP • 49, 55, 60, 201, 212, 227, 235, 271, 416
 GROUPS • 236, 280, 281, 417
 Graph pattern matching • 57
 Gregorian calendar • 24
 <general literal> • 387
 <general logarithm argument> • 336, 339
 <general logarithm base> • 336, 339
 <general logarithm function> • 335, 338, 339
 <general quantifier> • 246, 254, 255, 438, 472
 <general set function> • 217, 274, 275, 276

<general set function type> • 274
 generally depends on • 94
 generally includes • 94
 global object identifier • 31
 global unconditional singleton • 228
 globally identifiable • 31
 graph descriptor • 34
graph does not exist • 149, 204, 326, 332, 338, 348, 352, 357, 366, 452, 453
 graph element base type • 72
 <graph expression> • 6, 17, 120, 122, 129, 130, 131, 132, 144, 215, 218
 <graph initializer> • 17, 120, 121, 122
 <graph name> • 27, 144, 149, 284, 285, 412, 413
 <graph parameter definition> • 118, 120, 121, 122
 <graph parent specification> • 27, 144, 149, 284, 285
 <graph pattern> • 55, 56, 57, 60, 61, 63, 64, 65, 191, 216, 225, 226, 227, 230, 231, 233, 252, 306, 314, 435, 444, 467
 <graph pattern quantifier> • 60, 228, 239, 245, 246, 254, 255, 257, 259, 472, 489
 <graph pattern variable> • 60, 63, 412
 <graph pattern where clause> • 61, 62, 216, 225, 232, 233, 245, 306, 445, 468
 <graph reference> • 130, 151, 229, 252, 284
 <graph resolution expression> • 101, 284
 <graph source> • 144
 <graph specification> • 130, 147, 148
 graph type • 70
graph type conformance error • 145
graph type definition error • 161
graph type does not exist • 166, 312, 348, 358
 <graph type expression> • 6, 129, 131, 132, 145, 151
 <graph type initializer> • 151
 <graph type name> • 27, 151, 166, 287, 288, 412
 <graph type parent specification> • 151, 166, 287, 288
 <graph type reference> • 131, 132, 287
 <graph type resolution expression> • 287
 <graph type specification> • 131, 132, 153, 154
 <graph type specification body> • 153, 155, 156, 159, 160, 161, 162
 <graph variable> • 120, 121
 <graph variable declaration> • 118, 120, 121
 <graph variable definition> • 118, 120, 121, 122
 <grave accent> • 388, 392, 424, 425
 <greater than operator> • 278, 311, 418
 <greater than or equals operator> • 311, 312, 418, 419
 <group by clause> • 8, 45, 210, 211, 215, 271
 grouping • 211
 <grouping element> • 8, 211, 271
 <grouping element list> • 271
 grouping record • 213

HAVING • 215, 416
 HOME_GRAPH • 324, 325, 416
 HOME_PROPERTY_GRAPH • 324, 325, 416
 HOME_SCHEMA • 324, 325, 416
 <having clause> • 215
 <hex digit> • 389, 391, 392, 393, 395, 424, 426

— I —

IF • 103, 120, 121, 123, 124, 126, 127, 142, 143, 144, 145, 149, 151, 152, 166, 416
 IN • 203, 416
 INDEX • 52, 60, 203, 204, 205, 206, 417
 INSERT • 172, 416
 INT • 38, 67, 85, 398, 402, 403, 416
 INT128 • 67, 85, 398, 403, 416
 INT16 • 67, 85, 398, 403, 416
 INT256 • 67, 85, 398, 403, 416
 INT32 • 67, 85, 398, 403, 416
 INT64 • 67, 85, 398, 403, 416
 INT8 • 85, 398, 403, 416
 INTEGER • 38, 66, 67, 83, 85, 86, 398, 401, 402, 403, 404, 407, 416
 INTEGER128 • 67, 85, 86, 398, 403, 416
 INTEGER16 • 67, 85, 398, 403, 416
 INTEGER256 • 67, 85, 86, 398, 403, 416
 INTEGER32 • 67, 85, 86, 398, 403, 416
 INTEGER64 • 67, 85, 86, 398, 403, 416
 INTEGER8 • 85, 398, 403, 416
 INTERSECT • 185, 186, 416, 503
 IS • 240, 259, 260, 316, 317, 318, 319, 320, 329, 330, 331, 343, 349, 374, 416, 503
 <identifier> • 17, 55, 60, 173, 191, 203, 204, 205, 210, 211, 215, 242, 269, 299, 412, 413, 430, 431, 504
 <identifier extend> • 415, 421, 499, 500
 <identifier start> • 415, 420, 499
 immediately contain • 92
 <implementation-defined access mode> • 108
 inDegree • 336, 415
 <inDegree function> • 335, 336
 include • 94
 incoming working record • 45
 incoming working table • 45
 <independent value expression> • 274, 275, 276
 inessential • 93
 <inline procedure call> • 31, 168, 183, 193, 263, 264
 innermost • 92
 innermost executing command • 38
 innermost executing operation • 39
 innermost executing procedure • 38
 innermost executing procedure or command • 38
 innermost executing statement • 39
 <insert statement> • 47, 138, 172, 456, 504
 instant • 23

integer numbers • 67
 integer types • 67
 integers • 67
 interior variable • 237
 intermediate results • 93
 internal object identifiers • 31
invalid argument for natural logarithm • 340
invalid argument for power function • 340
invalid character value for cast • 377, 378, 380, 381, 385
invalid datetime function map key • 354, 355
invalid datetime function map value • 355
invalid duration format • 385
invalid duration function map key • 359
invalid number of paths or groups • 442
invalid reference • 48, 110, 186, 212, 221, 250, 267, 300, 455
invalid syntax • 44, 48, 161, 182, 267, 343, 344, 349, 350, 378, 379, 380, 381, 394
invalid time zone displacement value • 102
invalid transaction state • 41, 106, 112
invalid transaction termination • 109, 110
 is generally dependent on • 94
 <is label expression> • 163, 239, 240, 243, 250
 <is labeled or colon> • 319
 <is or colon> • 175, 179, 240, 504
 iterator variable • 60

— K —

KEEP • 51, 225, 226, 417
 <keep clause> • 51, 225, 226, 233, 468
 <key word> • 66, 320, 415, 422
 known not nullable • 88

— L —

LABEL • 163, 417
 LABELED • 319, 417, 503
 LABELS • 163, 417
 LAST • 277, 278, 417
 LEADING • 345, 347, 348, 350, 351, 416
 LEFT • 345, 346, 347, 416
 LENGTH • 335, 416
 LET • 199, 416
 LIKE • 131, 144, 145, 416
 LIKE_REGEX • 416
 LIMIT • 280, 416
 LIST • 399, 406, 416
 LN • 336, 340, 341, 416
 LOCALDATETIME • 87, 353, 354, 383, 384, 393, 399, 406, 416, 453
 LOCALTIME • 88, 353, 354, 382, 383, 384, 393, 399, 406, 416, 453
 LOCALTIMESTAMP • 353, 354, 416
 LOG • 336, 337, 416
 LOG10 • 336, 416

LOWER • 345, 349, 416
 lTrim • 345, 346, 347, 415
 <label conjunction> • 252, 253, 428, 472
 <label disjunction> • 163, 250, 252, 253, 428, 472
 <label expression> • 10, 59, 63, 163, 240, 252, 253, 259, 260,
 319, 428, 429, 438, 473, 503
 <label factor> • 252, 428
 <label name> • 59, 163, 175, 177, 178, 180, 252, 253, 257,
 260, 412, 413, 428, 473
 label name set • 163
 label namespace • 55
 <label negation> • 163, 250, 252, 253, 428, 473
 <label primary> • 252, 428
 <label set definition> • 155, 156, 157, 161, 163, 492
 <label set delimiter> • 175
 <label set specification> • 175, 179, 504
 <label term> • 252, 428
 <labeled predicate> • 309, 310, 319, 474, 502, 503
 <labeled predicate part 2> • 319, 373, 374
 <left angle bracket> • 257, 399, 419, 425, 426
 <left arrow> • 158, 241, 244, 249, 418, 419, 471
 <left arrow bracket> • 157, 240, 418, 419
 <left arrow tilde> • 241, 244, 257, 418, 419
 <left arrow tilde bracket> • 240, 418, 419
 left boundary variable • 237
 <left brace> • 111, 114, 153, 164, 175, 240, 254, 366, 368, 399,
 424, 425, 504
 <left bracket> • 102, 355, 365, 393, 425
 <left minus right> • 241, 244, 418, 419
 <left minus slash> • 256, 258, 418, 419
 <left paren> • 155, 158, 223, 239, 241, 249, 252, 257, 266,
 271, 274, 314, 322, 323, 329, 334, 335, 336, 345, 346, 353,
 357, 359, 363, 373, 376, 386, 397, 398, 399, 425, 471
 <left tilde slash> • 256, 258, 418, 419
 left to right • 58
 <length expression> • 335, 338
 <less than operator> • 278, 311, 312, 418, 419
 <less than or equals operator> • 311, 418, 419
 <let statement> • 29, 138, 199, 456
 library • 36
 library default procedure name • 37
 library procedure object • 37
 <like graph expression> • 131, 132
 <like graph expression shorthand> • 131, 132
 <limit clause> • 51, 206, 207, 215, 280
 <linear catalog-modifying statement> • 135, 140, 141
 <linear data-modifying statement> • 6, 55, 135, 169, 170
 <linear query expression> • 185, 187, 188
 <linear query statement> • 55, 115, 188, 189, 190
 list • 76
 <list concatenation> • 361
list data, right truncation • 361
 <list element> • 365, 394

list element error • 363
 <list element list> • 365, 394, 502
 <list literal> • 387, 390, 394, 396
 list of graph elements bound to ER • 446
 <list primary> • 361
 list type • 76
 <list value constructor> • 334, 365, 489, 502
 <list value constructor by enumeration> • 365, 390, 394,
 396
 <list value expression> • 203, 204, 327, 361, 363, 452, 489
 <list value expression 1> • 361
 <list value function> • 53, 361, 363, 502
 <list value type> • 397, 399
 <list value type name> • 365, 399
 <literal> • 16, 235, 324, 325, 377, 378, 379, 381, 382, 383,
 384, 385, 387, 394, 504
 local • 62
 <local binding table reference> • 290, 291
 <local datetime function> • 353, 354, 355
 <local function reference> • 296, 297
 <local graph reference> • 284, 285
 <local graph type reference> • 287, 288
 local namespace • 55
 <local procedure reference> • 293, 294
 <local time function> • 353, 354
 <lower bound> • 242, 245, 254, 438

— M —

MANDATORY • 136, 191, 194, 199, 204, 416
 MAP • 366, 399, 406, 416, 455
 MATCH • 48, 51, 191, 231, 247, 248, 314, 416
 MAX • 274, 416, 450
 MERGE • 416
 MESSAGE_TEXT • 417
 MIN • 274, 416, 450
 MOD • 335, 416
 MORE • 417
 MULTI • 223, 416
 MULTIPLE • 223, 416
 MUTABLE • 417
 <mandatory formal parameter list> • 223, 224
 <mandatory statement> • 29, 136, 138, 191, 194, 204, 456
 <mantissa> • 84, 379, 380, 390, 391, 395
 map • 78
 <map element> • 366, 367, 394
 <map element list> • 366, 367, 394
 <map key> • 354, 355, 359, 366, 367
 <map key type> • 399, 406
 <map literal> • 387, 390, 394, 396
 map type • 78
 <map value> • 355, 366, 367
 <map value constructor> • 26, 334, 353, 359, 366, 367

<map value constructor by enumeration> • 366, 367, 390, 394, 396
 <map value expression> • 327, 328
 <map value type> • 397, 399
 match • 231
 <match mode> • 65, 225, 227, 233, 467, 486
 <match statement> • 20, 138, 191, 215, 216, 306, 456
 material • 66
 <max length> • 397, 400, 401
 may cause • 57
 <min length> • 397, 400, 401
 minimum node count • 244
 minimum path length • 242
 <minus left bracket> • 157, 240, 241, 418, 419
 <minus sign> • 56, 240, 241, 244, 249, 257, 332, 333, 352, 357, 389, 394, 425, 426, 471
 <minus slash> • 241, 256, 258, 418, 419
 <modulus expression> • 335, 338
 most specific type • 65
 multi-path binding • 434
multiple assignments to a graph element property • 177
 multiply-declared-different-edges-matched • 227, 228
 multiply-declared-restrictive-path-mode • 236, 237
 multiset alternation operand • 242
 <multiset alternation operator> • 60, 239, 256, 415, 418

— N —

NEW • 416
 NFC • 317, 345, 347, 417
 NFD • 345, 417
 NFKC • 345, 417
 NFKD • 345, 417
 NODE • 7, 156, 406, 408, 417, 420
 NODES • 417
 NORMALIZE • 343, 345, 349, 416
 NORMALIZED • 317, 343, 349, 417
 NOT • 120, 121, 123, 124, 126, 127, 142, 144, 145, 151, 152, 311, 316, 317, 318, 319, 320, 329, 330, 374, 416, 501, 503
 NOTHING • 397, 416
 NULL • 316, 373, 374, 375, 377, 390, 393, 416, 501
 NULLIF • 373, 374, 416
 NULLS • 277, 278, 416
 NUMBER • 417
 NUMERIC • 416
 name • 60
 named graph descriptor • 35
 named graph type descriptor • 71
 <named procedure call> • 46, 56, 168, 183, 193, 263, 266
 named procedure descriptor • 50
 <natural logarithm> • 335, 336, 338, 340
 negative duration • 24
 <nested ambient data-modifying procedure specification> • 147

<nested data-modifying procedure specification> • 111, 112, 147, 148, 169, 170, 171, 218
 <nested graph query specification> • 120, 122, 131, 132, 147
 <nested graph type specification> • 131, 153
 <nested procedure specification> • 111, 112, 168, 183, 193, 264
 <nested query specification> • 114, 123, 125, 126, 128, 147, 189, 215, 218, 314, 372
 <newline> • 199, 393, 420, 421
 no data • 48, 457
 node • 253
 node base type • 72
 <node pattern> • 59, 61, 236, 237, 239, 242, 243, 244, 245, 248, 249, 250, 253, 437, 439, 471
 <node reference> • 320
 <node reference value expression> • 327
 <node reference value type> • 399, 400, 406
 node symbol binding • 431
 <node synonym> • 155, 399, 420
 node type • 72
 <node type definition> • 72, 74, 153, 154, 155, 161, 399, 406, 486
 <node type filler> • 155, 158, 159, 161
 <node type label set definition> • 155, 156
 <node type name> • 153, 154, 155, 156, 161
 <node type pattern> • 155
 <node type phrase> • 155
 <node type property type set definition> • 155, 156
 <node type reference> • 158, 159
 node variable • 245
 <non-delimited identifier> • 415, 421, 422, 474
 <non-delimiter token> • 19, 415, 421
 <non-parenthesized value expression primary> • 329, 330, 334, 373, 374
 <non-reserved word> • 415, 417
 <normal form> • 81, 163, 164, 317, 345, 347, 400, 401, 402, 404, 406
 <normalize function> • 81, 345, 347, 348, 349
 <normalized predicate> • 81, 309, 310, 317
 <normalized predicate part 2> • 317
 <not equals operator> • 311, 313, 329, 418, 419, 449, 450
 nothing type • 69
 <null literal> • 376, 387, 390, 393, 396
 <null ordering> • 277
 <null predicate> • 309, 310, 316, 501
 <null predicate part 2> • 316, 373
 null value not allowed • 326, 379, 381, 385
 nullability characteristic • 88
 nullable • 66
 <number of groups> • 235, 442
 <number of paths> • 234, 235, 442
 numbers • 67
 <numeric primary> • 332, 333

<numeric type> • 83, **397**, 402, 405
 numeric types • 67
 <numeric value expression> • 24, 275, 327, **332**, 333, 335, 336, 337, 338, 339, 340, 341, 346, 363
 <numeric value expression base> • **336**, 340
 <numeric value expression dividend> • **335**, 338
 <numeric value expression divisor> • **335**, 338
 <numeric value expression exponent> • **336**, 340
 <numeric value function> • 53, 332, **335**

— O —

OCCURRENCES_REGEX • 416
 OCTET_LENGTH • 335, 416
 OF • 131, 132, 144, 320, 397, 416
 OFFSET • 281, 416
 ON • 416
 ONLY • 108, 280, 281, 417
 OPTIONAL • 136, 172, 191, 196, 199, 204, 223, 416
 OR • 144, 146, 151, 152, 312, 329, 330, 375, 416
 ORDER • 273, 416
 ORDERED • 207, 416
 ORDINALITY • 52, 60, 203, 204, 205, 206, 417
 OTHERWISE • 185, 187, 416
 object base type name • 80
 <object name> • 302, **412**, 413
 object type • 65
 <octal digit> • 389, 391, **424**
 <of graph type> • 120, 121, 127, **131**, 132, 144, 145, 504
 <of type> • 123, 126, 131, 165, 223, **397**, 411, 504
 <of type signature> • **223**, 224
 <of value type> • **397**
 <offset clause> • 51, 206, 207, 215, **281**
 <offset synonym> • **281**
 omitted result • 47
 <open edge reference value type> • **399**, 406
 <open node reference value type> • **399**, 406, 408
 open record type • 406
 <optional formal parameter list> • **223**, 224
 <optional statement> • 29, 136, 138, 172, 191, **196**, 204, 456
 <order by and page statement> • 52, 138, **206**, 208, 456
 <order by clause> • 60, 206, 207, 215, **273**
 ordered • 35
 ordering operation • 450
 <ordering specification> • **277**
 <other digit> • **424**, 426
 <other language character> • **424**, **426**
 outDegree • 336, 415
 <outDegree function> • 335, **336**
 outermost • 92
 outgoing working record • 45
 outgoing working table • 45

— P —

PARAMETER • 101, 416
 PARTITION • 52, 206, 416
 PATH • 234, 235, 406, 416
 PATHS • 234, 235, 236, 416
 PATTERN • 417
 PATTERNS • 417
 POSITION_REGEX • 416
 POWER • 336, 338, 416
 PRECISION • 67, 87, 399, 404, 405, 416
 PROCEDURE • 51, 293, 416
 PROCEDURES • 416
 PRODUCT • 274, 416
 PROJECT • 217, 416
 PROPERTIES • 417
 PROPERTY • 120, 121, 131, 132, 144, 145, 146, 147, 151, 152, 153, 166, 417, 504
 <parameter> • 55, 103, **221**, 222, 304, 324
 <parameter cardinality> • **223**, 264, 267
 <parameter definition> • 101, **118**
 <parameter name> • 103, 120, 121, 123, 124, 126, 127, 221, 325, **412**, 413, 415
 parameter namespace • 55
 <parameter value specification> • **324**, 334
 <parameterized url path> • **299**, 300
 parameterized view • 52
 <parent catalog object reference> • 284, 285, 287, 288, 290, 291, 293, 294, 296, 297, **299**
 parent directory • 29
 <parent object relative url path> • **299**, 300
 <parenthesized boolean value expression> • **329**, 330
 <parenthesized formal parameter list> • **223**
 <parenthesized label expression> • **252**, 253, 428, 473
 <parenthesized path pattern expression> • 18, 60, 61, 62, 63, 64, 227, 228, 230, 231, 235, 236, 237, 239, **241**, 242, 245, 246, 247, 248, 249, 306, 430, 434, 435, 436, 437, 438, 442, 444, 445, 470, 472
 <parenthesized path pattern where clause> • 48, 61, 62, 216, 232, **241**, 245, 247, 249, 306, 442, 445, 472
 <parenthesized value expression> • **334**
 path binding • 431
 <path concatenation> • 48, 61, **239**, 242, 243, 244, 246, 248, 434, 436, 438, 440, 470
 <path factor> • **239**, 246, 436, 438, 440
 <path length expression> • **335**, 338
 <path mode> • 58, 63, 64, 227, **234**, 235, 236, 237, 238, 247, 434, 468
 <path mode prefix> • 226, **234**, 235, 236, 241, 247, 249, 434, 472
 <path multiset alternation> • 54, 60, 61, 62, 227, 228, 236, 237, **239**, 241, 242, 244, 246, 248, 440, 469
 <path or paths> • **234**, 235, 238, 469
 <path or subpath variable> • **412**, **413**

<path pattern> • 51, 57, 58, 61, 63, 64, 65, **225**, 226, 227, 228, 229, 230, 231, 233, 237, 238, 245, 247, 250, 413, 431, 435, 437, 441, 442, 445, 468, 492
 <path pattern expression> • 48, 56, 59, 225, 226, 227, 236, **239**, 241, 242, 243, 245, 247, 248, 249, 250, 436, 437, 440, 471
 <path pattern list> • **225**, 226, 227, 229
 <path pattern name> • **412**, 413
 <path pattern prefix> • 50, 51, 59, 225, 226, 227, 233, **234**, 235, 236, 468
 <path pattern union> • 50, 54, 61, 62, 64, 227, 228, 236, 237, **239**, 241, 242, 243, 244, 246, 248, 436, 440, 470
 path pattern union operand • 243
 <path primary> • **239**, 242, 243, 245, 246, 247, 248, 436, 438, 440, 470
 <path search prefix> • 58, 63, 64, 226, 233, **234**, 237, 441, 442, 468
 <path term> • **239**, 242, 243, 244, 246, 436, 438, 440
 path type • 77
 <path variable> • 225, 228, 229, 242, **413**
 <path variable declaration> • 59, 61, **225**, 226, 227, 233, 468
 path variable in the global scope • 229
 peers • 279
 <percent> • 252, **425**, **426**
 percentileCont • 274, 415
 percentileDist • 274, 415
 <period> • 84, 175, 179, 284, 287, 290, 293, 296, 299, 302, 371, 379, 380, 389, 391, 425, **426**, 504
 persistent • 88
 <plus sign> • 254, 255, 332, 333, 352, 357, 389, 425, **426**, 472
 possibly nullable • 88
 possibly variable length path pattern • 241
 <power function> • 335, **336**, 338, 340
 precede • 278
 <precision> • 83, **398**, 399, 401, 402, 403, 404, 405
 predefined • 66
 <predefined graph parameter> • 284, 285, **324**
 <predefined parameter> • **324**
 <predefined parent object parameter> • 299, 300, **324**
 <predefined schema parameter> • 282, **324**
 <predefined table parameter> • 290, 291, **324**
 <predefined type> • 9, 366, 376, **397**, 399, 400
 <predefined type literal> • **387**
 <predicate> • 56, 80, **309**, 310, 329
 preferred name • 66
 primary • 31
 primary element variable • 60
 primary object • 31
 primary variable • **17**, 245
 <primitive catalog-modifying statement> • **138**
 <primitive data-modifying statement> • **138**
 <primitive data-transforming statement> • **138**
 <primitive result statement> • 7, 55, 115, 135, 169, 170, 185, 189, **208**, 218

principal • 28
 <procedure argument> • **266**
 <procedure argument list> • **266**
 <procedure body> • 5, 53, 56, 92, 111, 112, **114**, **115**, 217, 220, 275
 <procedure call> • 54, 136, 168, 183, 193, 194, 196, **263**
procedure call error • 267
 procedure descriptor • 50
 procedure logic • 50
 <procedure name> • 293, 294, **412**, 413
 procedure object • 37
 <procedure parent specification> • **293**, 294
 <procedure reference> • 168, 183, 193, 266, **293**
 <procedure resolution expression> • **293**
 <procedure result type> • **223**, 224
 procedure signature • 50
 procedure signature descriptor • 51
 <procedure specification> • 100, **111**, 112, 113
 production rule • 89
 <project statement> • 7, 115, 136, 185, 208, **217**
 property graph • 33
 <property key value pair> • 173, 176, 177, 178, 191, **240**
 <property key value pair list> • 173, 175, 191, **240**
 <property name> • 165, 173, 175, 176, 177, 178, 179, 180, 192, 240, 371, **412**, 413, 504
 property namespace • 55
 <property reference> • 26, 62, 334, **371**, 473
 <property type definition> • 164, **165**, 475
 <property type definition list> • **164**, 165
 property type set • 164
 <property type set definition> • 62, 155, 156, 157, 162, **164**
 <property value type> • 65, 165, **410**
 pure function • 52

— Q —

QUERIES • 416
 QUERY • 51, 416
 <qualified binding table name> • **290**, 291
 <qualified function name> • **296**, 297
 <qualified graph name> • **284**, 285
 <qualified graph type name> • **287**, 288
 <qualified name prefix> • **302**
 <qualified object name> • 284, 285, 287, 288, 290, 291, 293, 294, 296, 297, **302**, 303
 <qualified procedure name> • **293**, 294
 <quantified path primary> • 61, 62, 228, **239**, 242, 243, 245, 246, 247, 248, 435, 436, 438, 440, 470
 <query conjunction> • **185**
 <query specification> • 63, **111**, 112, **114**, 193
 <query statement> • 54, 112, 114, **135**
 <question mark> • 60, 239, 257, 425, **426**
 <questioned path primary> • 60, **239**, 241, 242, 243, 245, 247, 248, 436, 438, 440, 470

<quote> • 388, 389, 391, 392, 425, **426**

— R —

RADIANS • 335, 339, **416**
 READ • 106, **108**, **417**
 REAL • 67, 87, 399, 405, **416**
 RECORD • 133, 368, 399, 406, 409, **416**
 RECORDS • 280, 281, **416**
 REFERENCE • 65, 80, 406, 408, 409, **416**
 RELATIONSHIP • 7, 162, 406, 409, 417, **420**
 RELATIONSHIPS • **417**, **420**
 REMOVE • 103, 179, **416**
 RENAME • **416**
 REPEATABLE • 65, 225, 233, **417**, **467**
 REPLACE • 144, 146, 151, 152, **416**
 REQUIRE • **416**
 RESET • **416**
 RESULT • **416**
 RETURN • 136, 172, 191, 199, 204, 210, 211, 314, **416**
 RETURNED_GQLSTATUS • **417**
 REVOKE • **416**
 RIGHT • 345, 346, 347, **416**
 ROLLBACK • 42, 109, **416**
 rTrim • 345, 346, 347, 348, **415**
 real approximate numeric type • 87, **405**
 record • 78
 <record literal> • **387**, **390**, 394, 395, 396
 <record type> • 66, 76, 78, 79, 165, 368, 369, 397, **399**, 406, 409, 411, 475
 <record value constructor> • **334**, **368**, 369, 370, 390, 394, 395, 396, 475
 <record value expression> • **327**, 328
 reduced match • 232
 reduced path binding • 432
 reference base type name • 80
 reference value • 38, 79
 <reference value expression> • **327**, **371**
 <reference value type> • 65, 79, 397, **399**, 400
 references • 306
 referent • **14**
 <refined edge reference value type> • 159, **399**, 406
 <refined node reference value type> • 155, **399**, 406, 408
 refined object type • 80
 regular approximate numeric type • 87, **405**
 regular decimal exact numeric type • 86, **404**
 <regular identifier> • 46, 250, 413, **415**, 499
 regular language of BNT • 437
 regular result • 47
 <relative url path> • **299**, 300, 304
 <remove item> • **179**, 180
 <remove item list> • **179**
 <remove label item> • **179**, 180
 <remove property item> • **179**, 180

<remove statement> • 19, 138, **179**, **456**
 <repeatable elements match mode> • **225**, 227, 233, **467**
 representation with reduced precision • **24**
 representative form • **421**
 <reserved word> • **415**, **422**
 restrictive • 236
 <result> • 47, **373**, 374, 375
 <result expression> • **373**, **375**
 <return item> • **210**, 211, 212, 214
 <return item alias> • **210**, 211
 <return item list> • **210**, 211
 <return statement> • 13, 45, 208, **210**, 211, 215, 275
 <return statement body> • **210**
 <reverse solidus> • 388, 389, 391, 392, 425, **426**
 <right angle bracket> • 257, 399, 418, 424, **425**
 <right arrow> • 158, 241, 244, 249, 418, **419**, **471**
 right boundary variable • 237
 <right brace> • 111, 114, 153, 164, 175, 240, 254, 366, 368, 399, **426**, **504**
 <right bracket> • 102, 355, 365, 393, **425**, **426**
 <right bracket minus> • 157, 240, 241, 418, **419**
 <right bracket tilde> • 157, 240, 418, **419**
 <right paren> • 155, 158, 223, 239, 241, 249, 252, 257, 266, 271, 274, 314, 322, 323, 329, 334, 335, 336, 345, 346, 353, 357, 359, 363, 373, 376, 386, 397, 398, 399, 425, **426**, **471**
 right to left • 58
 <rollback command> • 42, 107, **109**, **456**
 root directory • 28
 root schema • 28

— S —

SAME • 323, **416**
 SCALAR • **416**
 SCHEMA • 101, 142, 143, **416**
 SCHEMAS • **416**
 SCHEMATA • **416**
 SELECT • 203, 215, **416**
 SESSION • 101, 103, 104, 105, **416**
 SET • 101, 173, 175, **416**
 SHORTEST • 49, 227, 234, 235, 236, **417**
 SIGNED • 38, 67, 85, 86, 398, 402, 403, 404, **416**
 SIMPLE • 64, 228, 230, 234, 236, 238, **417**, 434, 468
 SIN • 335, 339, **416**
 SINGLE • 223, **416**
 SINH • 335, 339, **416**
 SKIP • 281, **416**
 SMALL • 67, 85, 86, 398, 402, 403, 404, **416**
 SMALLINT • 67, 85, 398, 402, 403, **416**
 SOURCE • 320, 321, **417**
 <SQL-interval literal> • **390**, 394
 SQRT • 336, **416**
 START • 106, **416**
 STARTS • **416**

STRING • 38, 66, 67, 81, 382, 383, 384, 397, 400, 407, 416,
 501
 SUBCLASS_ORIGIN • 417
 SUBSTRING • 66, 345, 346, 382, 383, 384, 416
 SUBSTRING_REGEX • 416
 SUM • 274, 416
 <same predicate> • 309, 310, 323, 474
 satisfied • 308
 satisfies • 428
 <scale> • 83, 398, 399, 402, 404, 405
 <schema name> • 142, 143, 282, 412
 <schema reference> • 32, 50, 101, 220, 282
 <search condition> • 59, 60, 61, 63, 64, 80, 172, 181, 198,
 199, 204, 215, 225, 232, 240, 241, 262, 308, 309, 343, 349,
 373, 375, 442, 444
 <searched case> • 373, 374, 375
 <searched when clause> • 373, 374, 375
 secondary • 31
 secondary object • 32
 <select graph match> • 215
 <select graph match list> • 215, 216
 <select item> • 215, 503
 <select item alias> • 215, 503
 <select item list> • 215
 <select query specification> • 215
 <select statement> • 13, 45, 115, 189, 215, 216
 <select statement body> • 215
 selective • 237
 <semicolon> • 425, 426
 separable • 433
 <separated identifier> • 412, 413
 <separator> • 46, 241, 332, 388, 389, 392, 393, 420, 421
 <session activity> • 100
 <session clear command> • 100, 104, 455
 <session close command> • 39, 100, 105, 455
 session command • 52
 session context • 39
 <session parameter> • 101
 <session parameter command> • 100
 session parameters • 31
 <session remove command> • 100, 103, 455
 <session set command> • 100, 101, 102
 <session set graph clause> • 101, 455
 <session set parameter clause> • 101, 455
 <session set schema clause> • 101, 455
 <session set time zone clause> • 101, 455
 session-modifying side effects • 57
 <set all properties item> • 175, 176, 177
 <set item> • 173, 175, 176, 177
 <set item list> • 173, 175
 <set label item> • 175, 176, 177, 178
 set of local matches to BNT • 437
 <set operator> • 185, 186

<set property item> • 175, 176, 177
 <set quantifier> • 185, 210, 215, 274, 275, 503
 <set statement> • 19, 138, 173, 175, 456
 <set time zone value> • 101, 102
 shall • 92
 <shortest path search> • 234, 235, 238, 443, 469
 <sign> • 332, 357, 389, 395
 signed 128-bit integer type • 85, 403
 signed 16-bit integer type • 85, 403
 signed 256-bit integer type • 85, 403
 signed 32-bit integer type • 85, 403
 signed 64-bit integer type • 85, 403
 signed 8-bit integer type • 85, 403
 signed big integer type • 85
 <signed binary exact numeric type> • 398, 402, 403
 <signed decimal integer> • 84, 389, 390
 signed exact numbers • 67
 signed exact numeric types • 67
 <signed numeric literal> • 377, 378, 387, 389, 395
 signed regular integer type • 85, 403
 signed small integer type • 85, 403
 signed user-specified integer type • 403
 signed user-specified integer types • 85
 simple • 58
 <simple Latin letter> • 424
 <simple Latin lower-case letter> • 422, 424, 427
 <simple Latin upper-case letter> • 422, 424, 427, 457
 <simple case> • 60, 373, 374
 <simple catalog-modifying statement> • 138, 140
 <simple comment> • 420, 421, 422, 423, 474, 475
 <simple comment character> • 420, 421
 <simple comment introducer> • 15, 241, 420
 <simple data-accessing statement> • 138, 169, 170
 <simple data-modifying statement> • 135, 138, 169, 170,
 218
 <simple data-reading statement> • 138
 <simple data-transforming statement> • 138
 <simple graph pattern> • 55, 56, 67, 172, 250, 251, 504
 <simple linear data-accessing statement> • 169, 218
 <simple linear query statement> • 169, 189, 218
 <simple path pattern> • 250, 251
 <simple path pattern list> • 250
 <simple query statement> • 138, 169, 170, 189
 <simple relative url path> • 299, 300
 <simple url path> • 299, 300, 301
 simple view • 52
 <simple when clause> • 373, 374
 <simplified concatenation> • 256, 258, 259
 <simplified conjunction> • 256, 257, 258, 259
 <simplified contents> • 256, 257, 258, 260
 <simplified defaulting any direction> • 56, 240, 248, 256,
 260, 470, 473
 <simplified defaulting left> • 248, 256, 260, 470, 473

<simplified defaulting left or right> • 256
 <simplified defaulting left or undirected> • 256
 <simplified defaulting right> • 248, 256, 260, 470, 473
 <simplified defaulting undirected> • 256
 <simplified defaulting undirected or right> • 256
 <simplified direction override> • 257, 258, 259, 260, 473
 <simplified factor high> • 256, 257
 <simplified factor low> • 256, 257, 259
 <simplified multiset alternation> • 256, 258, 259
 <simplified negation> • 257, 258, 260
 <simplified override any direction> • 56, 240, 257, 260, 261, 473
 <simplified override left> • 257, 259, 260, 261, 473
 <simplified override left or right> • 257, 260
 <simplified override left or undirected> • 257, 260
 <simplified override right> • 257, 260, 261, 473
 <simplified override undirected> • 257, 259
 <simplified override undirected or right> • 257, 260
 <simplified path pattern expression> • 47, 239, 248, 256, 257, 258, 470
 <simplified path union> • 256, 259
 <simplified primary> • 47, 257, 260
 <simplified quantified> • 257, 258, 259
 <simplified questioned> • 257, 258, 259
 <simplified secondary> • 257, 259
 <simplified term> • 256, 259
 <simplified tertiary> • 257, 259
 simply contained in • 92
 simply containing • 92
 simply contains • 92
 <single quoted character representation> • 388, 391
 <single quoted character sequence> • 387, 391, 421
 singleton degree of reference • 62
 site • 88
 <slash minus> • 241, 256, 258, 418, 419
 <slash minus right> • 256, 258, 418, 419
 <slash tilde> • 256, 258, 418, 419
 <slash tilde right> • 256, 258, 418, 419
 <solidus> • 282, 299, 332, 333, 357, 425, 426
 sort direction • 278
 <sort key> • 277, 278
 <sort specification> • 35, 277, 278, 279
 <sort specification list> • 22, 273, 277, 278, 450
 <source node type name> • 158, 159, 160, 161
 <source node type reference> • 157, 158, 160
 <source predicate part 2> • 320, 373, 374
 <source/destination predicate> • 60, 309, 310, 320, 321, 474
 <space> • 312, 378, 379, 380, 389, 392, 421, 424, 425
 specified by • 92
 specify • 92
 <square root> • 335, 336, 338
 stDev • 274, 415
 stDevP • 274, 415
 standard description • 49
 <standard digit> • 424, 426, 457, 502
 standard-defined classes • 457
 standard-defined features • 96
 standard-defined subclasses • 457
 start bracket symbol binding • 431
 <start position> • 345, 346, 348, 350
 <start transaction command> • 41, 100, 106, 456
 startNode • 415
 <statement> • 112, 114, 115, 116, 135
 <statement block> • 53, 56, 115
statement completion unknown • 27, 267, 333, 338, 339, 340, 341, 377, 378
 <statement mode> • 136, 191, 199, 203, 204
 static site • 32
 status • 46
 strict elements • 65
 strict interior variable • 237
 strict subtype • 21
 strict supertype • 21
 <string length> • 345, 346, 348, 350
 <string literal character> • 388, 392
 <string value expression> • 101, 102, 317, 327, 335, 342, 343, 489
 <string value function> • 47, 342, 345, 348, 502
 subpath symbol binding • 431
 <subpath variable> • 241, 242, 245, 413, 414
 <subpath variable declaration> • 60, 61, 241, 242, 245, 246, 249, 437, 472
substring error • 349, 351
 <substring function> • 345, 347, 348, 349
successful completion • 48, 457
successful outcome • 47
 symbols • 431
syntax error or access rule violation • 44, 48, 186, 212, 221, 250, 300, 394, 455

— T —

Feature T351, “Bracketed comments” • 503
 TABLE • 123, 133, 290, 417
 TABLES • 417
 TAN • 335, 339, 416
 TANH • 335, 339, 416
 THEN • 115, 373, 374, 375, 416
 TIES • 49, 52, 206, 235, 280, 281, 417
 TIME • 88, 101, 353, 354, 383, 390, 393, 399, 406, 416, 453
 TIMESTAMP • 390, 416
 TO • 158, 417
 TRAIL • 64, 227, 230, 234, 236, 238, 417, 434, 468
 TRAILING • 345, 347, 348, 350, 351, 416
 TRANSACTION • 106, 417
 TRANSLATE_REGEX • 416

TRIM • 345, 346, 347, 348, 363, 416
 TRUE • 329, 380, 381, 387, 395, 416
 TRUNCATE • 416
 TYPE • 131, 132, 145, 149, 151, 152, 153, 155, 157, 166, 287,
 417
 TYPES • 417
 tail • 415
 <temporal literal> • 387, 390
 <temporal type> • 397, 399
 temporary • 88
 <term> • 332, 357, 358
 <then statement> • 115, 116, 117
 <tilde> • 158, 241, 244, 257, 425, 426
 <tilde left bracket> • 157, 240, 418, 419
 <tilde right arrow> • 241, 244, 418, 419
 <tilde slash> • 256, 258, 418, 420
 time • 23
 <time function> • 353, 354
 <time function parameters> • 353, 354, 355
 <time literal> • 390, 393, 395
 time of day • 24
 time scale • 24
 time shift • 24
 <time string> • 353, 355, 390, 393, 395, 418
 toLower • 345, 415
 toUpper • 345, 415
 <token> • 46, 241, 415, 421
 trail • 58
 <transaction access mode> • 108
 <transaction activity> • 100, 112
 <transaction characteristics> • 106, 108, 483
 transaction command • 52
 <transaction mode> • 108
transaction resolution unknown • 27, 382, 383, 384
transaction rollback • 27, 110
 transaction-modifying side effects • 57
 transient • 88
 <trigonometric function> • 335, 338, 339
 <trigonometric function name> • 335, 339
 <trim byte string> • 346, 351
 <trim character string> • 345, 348, 350, 502
trim error • 350, 351
 <trim function> • 345, 347, 348, 350, 502
 <trim list function> • 363
 <trim source> • 345, 347, 348, 350, 502
 <trim specification> • 345, 346, 350, 351, 502
 <truth value> • 329, 330
 truth values • 66
 type hierarchy • 68
 type object • 37
 <type signature> • 8, 53, 223, 224
 type signature descriptor • 51

— U —

UBIGINT • 67, 86, 398, 402, 404, 416
 UINT • 67, 86, 398, 402, 404, 416
 UINT128 • 67, 86, 398, 403, 416
 UINT16 • 67, 85, 398, 403, 416
 UINT256 • 67, 86, 398, 403, 416
 UINT32 • 67, 86, 398, 403, 416
 UINT64 • 67, 86, 398, 403, 416
 UINT8 • 85, 398, 403, 416
 UNDIRECTED • 158, 159, 160, 417
 UNION • 185, 186, 416, 503
 UNIQUE • 416
 UNIT • 416
 UNIT_BINDING_TABLE • 324, 325, 416
 UNIT_TABLE • 324, 325, 416
 UNKNOWN • 329, 387, 395, 416
 UNNEST • 203, 416
 UNSIGNED • 67, 85, 86, 398, 402, 403, 404, 416
 UNWIND • 416
 UPPER • 345, 349, 416
 USE • 218, 416
 USMALLINT • 67, 86, 398, 402, 404, 416
 unbounded quantifier • 255
 <unbroken accent quoted character sequence> • 388, 391,
 420, 421
 <unbroken character string literal> • 387, 390, 391, 393,
 394, 395
 <unbroken double quoted character sequence> • 387, 388,
 391
 <unbroken single quoted character sequence> • 387, 388,
 391
 unconditional singleton scope index • 228
 <underscore> • 389, 390, 391, 422, 425, 426
 undirected • 58
 <unicode 4 digit escape value> • 389
 <unicode 6 digit escape value> • 389
 <unicode escape value> • 388, 389, 392, 500
 union type • 69
 unit binding table • 35
 unit binding table type • 76
 unit record • 79
 unit record type • 79
 unordered • 35
 unsigned 128-bit integer type • 86, 403
 unsigned 16-bit integer type • 85, 403
 unsigned 256-bit integer type • 86, 403
 unsigned 32-bit integer type • 86, 403
 unsigned 64-bit integer type • 86, 403
 unsigned 8-bit integer type • 85, 403
 unsigned big integer type • 86, 404
 <unsigned binary exact numeric type> • 398, 402, 403
 <unsigned binary integer> • 389, 391

<unsigned decimal integer> • 84, **389**, 390, 391, 397, 398
 unsigned exact numbers • 67
 unsigned exact numeric types • 67
 <unsigned hexadecimal integer> • 378, 380, **389**, 391
 <unsigned integer> • 84, 235, 254, 324, 379, 380, **389**, 391, 395
 <unsigned integer specification> • 64, 234, 235, 280, 281, **324**, 325, 326
 <unsigned literal> • 324, **387**
 <unsigned numeric literal> • 378, 387, **389**, 415
 <unsigned octal integer> • **389**, 391
 unsigned regular integer type • 86, 404
 unsigned small integer type • 86, 404
 unsigned user-specified integer type • 404
 unsigned user-specified integer types • 86
 <unsigned value specification> • **324**, 325, 334
 <upper bound> • 246, **254**, 255, 438, 472
 <url path parameter> • 27, 149, 166, 282, 284, 287, 290, 293, 296, 299, **304**
 <url segment> • 17, **299**, 301
 <use graph clause> • 43, 54, 169, 189, **218**, 219
 user-defined • 66
 user-specified approximate numeric type • 405
 user-specified approximate numeric types • 87
 user-specified decimal exact numeric type • 404
 user-specified decimal exact numeric types • 86

— V —

VALUE • 118, 119, **126**, 372, 416
 VALUES • 416
 VARBINARY • 67, 82, 397, 401, 416, 501
 VARCHAR • 38, 67, 81, 397, 400, 416, 501
 VERTEX • 7, 156, 406, 408, 417, 420
 VERTICES • 417
 <value expression> • 7, 53, 55, 56, **118**, 126, 128, 173, 175, 176, 177, 181, 192, 201, 210, 211, 215, 217, 240, 250, 266, 274, 275, 277, 311, 312, **327**, 334, 365, 366, 370, 373, 375, 376, 377, 394, 444
 <value expression primary> • 45, 316, 327, 332, **334**, 342, 352, 357, 361
 <value initializer> • 17, **126**, 127, 128
 <value name> • **412**, 413
 <value parameter definition> • 118, **126**, 127, 128
 <value query expression> • 26, 334, **372**
 <value specification> • **324**, 325, 326, 490
 <value type> • 14, 65, 126, 127, 223, **397**, 399, 400, 406, 410, 411, 504
 <value variable> • 118, **126**, 127, 201
 <value variable declaration> • 118, **126**, 127
 <value variable definition> • 8, 118, **126**, 127, 128
 <variable name> • 269, 270, 412, **413**
 variable quantifier • 255
 <verbose binary exact numeric type> • 398

<vertical bar> • 59, 60, 239, 252, 256, 425, **426**
 view • 52

— W —

WALK • 64, 234, 235, 236, 238, 417, 434, 468
 WHEN • 172, 373, 374, 375, 417
 WHERE • 4, 192, 198, 204, 225, 240, 241, 247, 262, 417, 444
 WITH • 49, 52, 60, 203, 204, 205, 206, 235, 280, 281, 417
 WITHOUT • 417
 WRITE • 106, 108, 417
warning • 48, 145, 149, 152, 166, 379, 381, 385, 457
 <when clause> • 172, 181
 <when operand> • 373, 374
 <when operand list> • 373, 374
 <where clause> • 52, 136, 171, 172, 175, 179, 181, 194, 196, 198, 199, 201, 203, 204, 206, 210, 215, 216, 217, **262**
 <whitespace> • 343, 350, 377, 378, 379, 381, 382, 383, 384, 385, **420**, 421
 <wildcard label> • 59, **252**, 253, 428, 473
 with an intervening instance of • 92
 without an intervening instance of • 92
 word • 431
 working graph • 43
 working object • 43
 working schema • 43

— X —

XOR • 329, 330, 417

— Y —

YIELD • 229, 269, 417
 <yield clause> • 115, 116, 117, 225, 228, 229, 245, 266, 267, **269**, 270
 <yield item> • 229, **269**, 270
 <yield item alias> • **269**, 270
 <yield item list> • 229, **269**
 <yield item name> • **269**, 270

— Z —

ZERO • 417
 ZONE • 101, 417
 zero-node path • 7

Editor's Notes

Some possible problem and language opportunities have been observed with the specifications contained in this document. Further contributions to this list are welcome. Deletions from the list (resulting from change proposals that correct the problems or from research indicating that the problems do not, in fact, exist) are even more welcome.

Because of the dynamic nature of this list (problems being removed because they are solved, new problems being added), each problem or opportunity has been assigned a "fixed" number. These numbers do not change from draft to draft.

Possible Problems: Major Technical

[GQL-000] The following Possible Problem has been noted:

Severity: major technical

Reference:

Note At: None.

Source: Your humble Editors.

Possible Problem:

In the body of the Working Draft, there occasionally appears a point that requires particular attention, highlighted thus:

** Editor's Note (number 284) **

Text of the problem.

Solution:

None provided with comment.

[GQL-006] The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 20.2, "<value type>".

Note At: Editor's Note number 258.

Source: Editors.

Possible Problem:

The collection data types needed by GQL need to be defined.

Solution:

None provided with comment.

[GQL-007] The following Possible Problem has been noted:

Severity: major technical

Reference: The chosen type checking modes are expected to have global impact on the document.

« WG3:BER-040R3 »

« WG3:BER-094R1 »

Note At: Editor's Note number 37, Editor's Note number 57, Editor's Note number 102, Editor's Note number 249, Editor's Note number 95, Editor's Note number 98.

Source: Editors.

Possible Problem:

The type checking mode or modes and applicable coercions of GQL need to be defined.

Many of the rules derived from SQL make use of the declared and/or specific data types of BNF terms, results, etc. These rules will need adjustment when the type checking mode or modes and system of GQL is properly defined and suitable for both schema-free and schema-full scenarios. It may also be appropriate, to consider separating Type Rules from Static Rules and General Rules.

« WG3:BER-026 deleted a reference to LO GQL-022 »

Solution:

None provided with comment.

GQL-008 The following Possible Problem has been noted:

Severity: major technical

Reference: Clause 19, "Value expressions".

Note At: Editor's Note number 222.

Source: Editors.

Possible Problem:

The graph-type specific expressions needed by GQL need to be defined.

Solution:

None provided with comment.

GQL-018 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.12, "Data types", Subclause 4.16.3, "Predefined value types".

Note At: Editor's Note number 30, Editor's Note number 224.

Source: Editors.

Possible Problem:

The null value in GQL is currently not included in every data type but treated separately. This has not yet been reflected fully in the document and requires further discussion.

Solution:

None provided with comment.

GQL-021 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.10.2.3, "Procedure signatures", Subclause 4.10.2.5, "Type signature descriptor", Subclause 16.14, "<inline procedure call>".

Note At: Editor's Note number 13, Editor's Note number 14, Editor's Note number 63, Editor's Note number 173.

Source: Editors.

Possible Problem:

Procedure signatures and type signatures need to be explicitly declared or inferred syntactically, and verified by procedure calls and invocations.

Solution:

None provided with comment.

GQL-050 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 16.7, "<path pattern expression>".

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Note At: Editor's Note number 164.

Source: W04-009R1.

Possible Problem:

WG3:W04-009R1 defined "effectively bounded group variable" but did not use the definition. The definition will be used when we define predicates on aggregates, at which time we will want a Syntax Rules stating that if a group variable *GV* is referenced in a WHERE clause, then it shall be effectively bounded, and the reference shall be contained in an aggregated argument of an <aggregate function>.

Was linked to PP PGQ-044 but that was resolved for PGQ by W20-030.

Solution:

None provided with comment.

GQL-061 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.13, "Type hierarchy outline".

Note At: Editor's Note number 33, Editor's Note number 34, Editor's Note number 35, Editor's Note number 36, Editor's Note number 38, Editor's Note number 39.

Source: W05-020, W12-029.

Possible Problem:

W05-020 suggests that the text in this Subclause has not been considered by WG3 and that consensus agreement is required.

Solution:

None provided with comment.

GQL-062 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.14, "Graph types".

Note At: Editor's Note number 40, Editor's Note number 41, Editor's Note number 43.

Source: W05-020.

Possible Problem:

W05-020 suggests that although the contents of this Subclause have been discussed, consensus has not yet been reached on all aspects. It has been touched on in [W04-013] and [MMX-079r1] and is part of the roadmap in [W02-014]. Consensus agreement is required.

Solution:

None provided with comment.

GQL-063 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.15, "Binding table types".

Note At: Editor's Note number 47.

Source: W05-020.

Possible Problem:

W05-020 suggests that despite being touched on in [W04-013], further discussion needed to establish consensus agreement.

Solution:

None provided with comment.

GQL-064 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.16.1, "Constructed types".

Note At: Editor's Note number 48.

Source: W05-020.

Possible Problem:

W05-020 suggests that despite being discussed in [W03-034] with the exception of Subclause 4.16.1.3, "Path type", which is included in the roadmap in [W02-014] and touched in in [W04-013], further discussion is needed to establish consensus agreement.

Solution:

None provided with comment.

GQL-066 The following Possible Problem has been noted:

Severity: major technical

Reference: Clause 9, "Procedures".

Note At: Editor's Note number 62, Editor's Note number 64.

Source: W05-020.

Possible Problem:

W05-020 suggests that the text in this clause has not been considered by WG3 and that consensus agreement is required.

WG3:RKE-015 has reviewed and reached consensus agreement for Subclause 9.3, "<procedure body>".

Solution:

GQL-067 The following Possible Problem has been noted:

Severity: major technical

Reference: Clause 10, "Variable and parameter declaration and definition".

Note At: Editor's Note number 65.

Source: W05-020.

Possible Problem:

W05-020 suggests that the text in this clause has not been considered by WG3 and that consensus agreement is required.

Solution:

None provided with comment.

GQL-068 The following Possible Problem has been noted:

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Severity: major technical

Reference:

« WG3:BER-040R3 »

Clause 11, “Object and object type expressions”.

Note At: Editor's Note number 68.

Source: W05-020.

Possible Problem:

W05-020 suggests that, although the text in some subclauses of this clause is not likely to be controversial and indeed some of the text is based on accepted proposals, further discussion is needed to establish consensus agreement. Specifically, W05-020 said:

- Subclause 11.2, “<graph expression>”, is based on [MMX-028r2] and [MMX-055] but further discussion is needed.
- Subclause 11.3, “<graph type expression>”, is based on [MMX-028r2] and [MMX-055] but further discussion is needed.

Solution:

None provided with comment.

GQL-069 The following Possible Problem has been noted:

Severity: major technical

Reference: Clause 12, “Statements”.

Note At: Editor's Note number 69.

Source: W05-020.

Possible Problem:

W05-020 suggests that the text in this clause, which was outlined in BNE-023, needs further discussion to achieve consensus agreement.

Solution:

None provided with comment.

GQL-071 The following Possible Problem has been noted:

Severity: major technical

Reference: Clause 14, “Data-modifying statements”.

Note At: Editor's Note number 86.

Source: W05-020.

Possible Problem:

W05-020 suggests that, although the text in the subclauses of this clause have been discussed in various papers further discussion is needed to establish consensus agreement. Specifically, W05-020 said:

- Discussion is needed for Subclause 14.1, “<linear data-modifying statement>”.

« WG3:BER-049 deleted deleted one item »

- Discussion is needed for Subclause 14.2, “<do statement>”.

- Subclause 14.3, “<insert statement>”, was discussed in [W06-009] and [BNE-023] but further discussion is needed.

« WG3:BER-050 deleted one item »

- Subclause 14.4, “<set statement>”, was discussed in [W06-009] and [BNE-023] but further discussion is needed.
- Subclause 14.5, “<remove statement>”, was discussed in [W06-009] and [BNE-023] but further discussion is needed.
- Subclause 14.6, “<delete statement>”, was discussed in [W06-009] and [BNE-023] but further discussion is needed.
- Subclause 14.7, “<call data-modifying procedure statement>”, was discussed in [BNE-023] but further discussion is needed.

Solution:

None provided with comment.

GQL-072 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 15.7.1, “<primitive result statement>”.

Note At: Editor's Note number 124.

Source: W05-020.

Possible Problem:

W05-020 suggests that discussion of this Subclause is needed to establish consensus.

Solution:

None provided with comment.

GQL-073 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 15.7.4, “<project statement>”.

Note At: Editor's Note number 135.

Source: W05-020.

Possible Problem:

W05-020 suggests that discussion of this Subclause is needed to establish consensus.

« WG3:BER-081 »

It needs to be specified in the SRs of <project statement>, which data types are acceptable result types for the <project statement>. More specifically, this can happen by restricting the declared type of the immediately contained <value expression>.

Solution:

None provided with comment.

GQL-075 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 16.3, “Named elements”.

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Note At: Editor's Note number 137.

Source: W05-020.

Possible Problem:

W05-020 suggests that discussion of this Subclause is needed to establish consensus.

Solution:

None provided with comment.

GQL-076 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 16.4, “<type signature>”.

Note At: Editor's Note number 138, Editor's Note number 139.

Source: W05-020.

Possible Problem:

W05-020 suggests that discussion of this Subclause is needed to establish consensus.

Solution:

None provided with comment.

GQL-078 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 16.17, “<group by clause>”.

Note At: Editor's Note number 184.

Source: W05-020, BER-019.

Possible Problem:

W05-020 suggests that discussion of this Subclause is needed to establish consensus.

Specific issue 1: The <group by clause> uses the <grouping element> as <binding variable> for defining a new binding variable. In most other parts of the draft <binding variable> is only used when reading a presumably already defined variable. While places like <value variable definition> use <binding variable name> (and not <binding variable>) when defining a new variable.

Specific issue 2: The [Syntax Rule 1](#)) "Every <binding variable> shall unambiguously reference a column of the current working table." seems not to make any sense.

Solution:

Specific issues 1: Replacing <binding variable> with <binding variable name> in the format of <grouping element> should be the kernel of a sufficient solution.

Specific issues 2: Remove the SR.

GQL-083 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 19.16, “<map value constructor>”, Subclause 19.2, “<value expression>”.

« [WG3:BER-082R1](#) »

Note At: Editor's Note number 245, Editor's Note number 223.

Source: W05-020.

Possible Problem:

W05-020 suggests that discussion of this Subclause is needed to establish consensus.

Solution:

None provided with comment.

GQL-085 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 19.19, “<property reference>”.

Note At: Editor's Note number 247.

Source: W05-020.

Possible Problem:

W05-020 suggests that discussion of this Subclause is needed to establish consensus.

Solution:

None provided with comment.

GQL-086 The following Possible Problem has been noted:

Severity: major technical

Reference: Clause 20, “Lexical elements”.

Note At: Editor's Note number 253.

Source: W05-020.

Possible Problem:

Clause 20, “Lexical elements”, is based on Clause 5, “Lexical elements”, of SQL/Foundation and adapted by [MMX-011r1], [MMX-028r2] and [W03-034]. Two proposals, [W04-015] and [W04-019], attempted to resolve perceived issues with this clause and both were rejected by WG3. W05-020 suggests that further discussion of this clause is needed to establish consensus.

W12-029 agreed on the following parts of Subclause 20.2, “<value type>”:

- The Format of <predefined type>.
- The Syntax Rules 1-42) introduced by W12-029.
- The General Rules 1-6) introduced by W12-029.

Solution:

None provided with comment.

GQL-087 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 21.11, “Result of value type combinations”.

Note At: Editor's Note number 278.

Source: W05-020.

Possible Problem:

W05-020 suggests that discussion of this Subclause is needed to establish consensus.

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Solution:

None provided with comment.

GQL-158 The following Possible Problem has been noted:

Severity: major technical

Reference: Numerous Syntax Rules and General Rules.

[« Editorial »](#)

Note At: Editor's Note number 75, Editor's Note number 78, Editor's Note number 84, Editor's Note number 85, Editor's Note number 103, Editor's Note number 104, Editor's Note number 109, Editor's Note number 110, Editor's Note number 183, Editor's Note number 199, Editor's Note number 200, Editor's Note number 201, Editor's Note number 202, Editor's Note number 204, Editor's Note number 205, Editor's Note number 206, Editor's Note number 207, Editor's Note number 208, Editor's Note number 209, Editor's Note number 211, Editor's Note number 212, Editor's Note number 213, Editor's Note number 221.

Source: Editors.

Possible Problem:

Exception conditions to be raised by Syntax Rules, General Rules, and during processing in general need to be determined for each exceptional situation that may occur.

Solution:

Consistently raise exception conditions.

GQL-164 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 16.5, “<graph pattern>”, Subclause 16.7, “<path pattern expression>”, Subclause 16.9, “<label expression>”, Subclause 20.7, “<GQL terminal character>”.

Note At: Editor's Note number 140, Editor's Note number 141, Editor's Note number 144, Editor's Note number 145, Editor's Note number 146, Editor's Note number 148, Editor's Note number 149, Editor's Note number 150, Editor's Note number 151, Editor's Note number 162, Editor's Note number 166, Editor's Note number 167, Editor's Note number 267, Editor's Note number 265.

Source: Editors.

Possible Problem:

Differences between Syntax Rules and General Rules shared between SQL/PGQ and GQL should be minimized.

See the different applications of W13-018 in SQL/PGQ and GQL for more places that should be considered.

Solution:

None provided with comment.

GQL-165 The following Possible Problem has been noted:

Severity: major technical

Reference:

[« WG3:BER-050 »](#)

Subclause 14.3, “<insert statement>”, Subclause 14.6, “<delete statement>”.

| **Note At:** Editor's Note number 92, Editor's Note number 101.

Source: Editors.

Possible Problem:

All omitted or incomplete General Rules regarding the modification of GQL-objects by data-modifying statements need to be fully specified.

Solution:

None provided with comment.

[GQL-170] The following Possible Problem has been noted:

Severity: major technical

Reference: Annex G, "Graph serialization format".

Note At: Editor's Note number 283.

Source: Editors.

Possible Problem:

It needs to be decided, if GQL should sketch a simple graph serialization format, e.g., by re-using (parts of) the syntax and semantics of Subclause 14.3, "<insert statement>", and Subclause 13.4, "<create graph statement>".

Solution:

None provided with comment.

[GQL-172] The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.10.4.6, "Scope of names", Subclause 16.19, "<aggregate function>", Subclause 15.7.2, "<return statement>".

Note At: Editor's Note number 15, Editor's Note number 192, Editor's Note number 129, Editor's Note number 163.

Source: Editors.

Possible Problem:

Rules regarding the scope and namespaces in which identifiable constructs (such as variables) are valid need to be defined.

Solution:

None provided with comment.

[GQL-173] The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.12.1, "General introduction to data types", Subclause 21.8, "Equality operations".

Note At: Editor's Note number 31.

Source: Editors.

Possible Problem:

Equality rules need to be specified.

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Solution:

None provided with comment.

GQL-174 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.12.1, "General introduction to data types", Subclause 21.9, "Ordering operations", Subclause 4.16.1.5, "Record type", Subclause 18.3, "<comparison predicate>".

Note At: Editor's Note number 32, Editor's Note number 51, Editor's Note number 277, Editor's Note number 50, Editor's Note number 219.

Source: Editors.

Possible Problem:

Rules regarding the comparison and sorting of values need to be specified.

Solution:

None provided with comment.

GQL-178 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 15.7.2, "<return statement>".

Note At: Editor's Note number 130.

Source: Editors.

Possible Problem:

Equivalence (equality relation that treats nulls as identical, e.g., when using DISTINCT) to be specified.

Solution:

None provided with comment.

GQL-179 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.2.5, "GQL-catalog".

Note At: Editor's Note number 5.

Source: Editors.

Possible Problem:

Descriptors of all objects in the GQL-catalog need to be defined.

Solution:

WG3:BER-061 defined the descriptors for GQL-directory and GQL-schema.

GQL-187 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 15.7.3, "<select statement>".

Note At: Editor's Note number 132.

Source: Editors.

Possible Problem:

The <select statement> needs to be fully specified.

Solution:

Syntax transformation to a <return statement>.

GQL-190 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 20.6, "<token> and <separator>".

Note At: Editor's Note number 254.

Source: Editors, WG3:W17-027.

Possible Problem:

The escaping syntax used for <delimited identifier>s needs to be decided.

Solution:

None provided with comment.

GQL-192 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.2.5.3, "GQL-schemas".

Note At: Editor's Note number 6, Editor's Note number 8.

Source: Editors.

Possible Problem:

The mechanisms required for referencing named and unnamed descriptors need to be fully specified.

Solution:

None provided with comment.

GQL-198 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 4.16.3.4.5, "Approximate numeric types".

Note At: Editor's Note number 54.

Source: Editors.

Possible Problem:

Conformance features of GQL need to be fully specified.

Solution:

None provided with comment.

GQL-202 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 19.20, "<value query expression>".

Note At: Editor's Note number 250.

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Source: Editors.

Possible Problem:

The semantics [Subclause 19.20, “<value query expression>”](#), have not been defined. The appropriate Syntax and General Rules must be defined.

Solution:

None provided with comment.

GQL-205 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 16.19, “<aggregate function>”.

Note At: Editor's Note number 189.

Source: Editors.

Possible Problem:

It is strongly suspected that the Syntax and General Rules of [Subclause 16.19, “<aggregate function>”](#), are incomplete. These rules must be reviewed and completed as necessary.

Solution:

None provided with comment.

GQL-215 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 20.2, “<value type>”.

Note At: Editor's Note number 257.

Source: W12-029.

Possible Problem:

Support for graph types and graph element types with and without attached schema information needs to be developed further.

Solution:

None provided with comment.

GQL-216 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 20.2, “<value type>”.

Note At: Editor's Note number 259, Editor's Note number 260.

Source: W12-029.

Possible Problem:

Use of 1-based numbering of the characters of character strings and the bytes of byte strings is in alignment with SQL but deviates from the convention of using 0-based numbering commonly adopted by virtually any other modern mainstream computer language.

It needs to be decided if and how this issue is to be addressed.

Solution:

None provided with comment.

GQL-251 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 16.7, “<path pattern expression>”.

Note At: Editor's Note number 158.

Source: Email from Petra Selmer 2022-05-09 17:41. (See W21-009 Seq#519,P16-USA-293.)

Possible Problem:

The syntax for <abbreviated edge pattern> is incompatible with syntax in the widely used graph query language Cypher. The following pattern:

(a) --> (b)

represents a single directed edge pattern from the node a to the node b in Cypher, but would represent a <simple comment introducer> and comment text in SQL.

Solution:

None provided with comment.

GQL-253 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 13.4, “<create graph statement>”.

Note At: Editor's Note number 76, and Editor's Note number 77.

Source: WG3:W21-059.

Possible Problem:

Ballot comment WG3:W21-010 P00-USA-23 noted the following problems in Subclause 13.4, “<create graph statement>”:

- The Format for a graph label set definition is not yet defined.
- The Format for a graph property set definition is not yet defined.

Solution:

None provided with comment.

GQL-276 The following Possible Problem has been noted:

Severity: major technical

Reference: Subclause 17.2, “Graph references”.

Note At: Editor's Note number 198.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result. Subclause 17.2, “Graph references” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

Possible Problems: Minor Technical

[GQL-038] The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 4.14.3.4, "Node type descriptor", Subclause 4.14.3.7, "Edge type descriptor".

Note At: Editor's Note number 46, Editor's Note number 45, Editor's Note number 80, Editor's Note number 81.

Source: Email Hannes Voigt, 2020-08-20 15:39.

Possible Problem:

These sections assume the node/edge type name is part of the node/edge type (descriptor). This assumption has not been part of the original thinking behind MMX-028r2. The consequences of this change have not been fully discussed, particularly the consequence on the definitions of: When two node/edge type descriptors describe equal node/edge types When a node/edge type is considered a subtype of another node/edge type When does the name matter with regards to assignability of values to sites of a given named type, or when not.

Solution:

None provided with comment.

[GQL-182] The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 20.1, "<literal>".

Note At: Editor's Note number 60, Editor's Note number 241, Editor's Note number 255.

Source: Discussion of W08-014, see W09-001.

Possible Problem:

During the discussion of W08-014, see W09-001, it was apparent that there was no consensus on the best way to specify a time zone. Should the standard allow both an explicit offset and an IANA time zone designator and should the combination also be allowed. This topic needs to be revisited to establish consensus.

Solution:

None provided with comment.

[GQL-189] The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 21.8, "Equality operations".

Note At: Editor's Note number 273.

Source: Editors.

Possible Problem:

The prohibitions and restrictions by value type on operations that involve testing for equality need to be fully specified.

Solution:

None provided with comment.

GQL-200 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 17.7, “<catalog object reference>”.

Note At: Editor's Note number 210.

Source: Editors.

Possible Problem:

There are differences between the set of allowed characters in an <url segment>. A more detailed analysis is required to better align URL segments and <identifier>s.

Solution:

A possible option might be to define how <identifier>s are to be mapped implicitly to <url segment>s.

Alternatively, <url segment> could be redefined without reference to <identifier> in order to align it with URL standards.

GQL-232 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 4.3.4.1, “Introduction to graphs”.

Note At: Editor's Note number 7.

Source: W13-018.

Possible Problem:

It has been questioned whether the term “label set” might give the wrong impression, that any set of labels constitutes a label set, whereas it is only those sets of labels designated in the construction of a graph that qualify as label sets. Some suggestions for alternative terms are “label combination”, “label grouping” and “labeling”.

Was linked to SQL/PGQ PP PQG-068, which was resolved for PGQ only by W19-013.

Solution:

None provided with comment.

GQL-254 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 10.2, “Graph variable and parameter declaration and definition” and Subclause 10.4, “Value variable and parameter declaration and definition”.

Note At: Editor's Note number 66 and Editor's Note number 67.

Source: WG3:W21-044.

Possible Problem:

Subclause 10.2, “Graph variable and parameter declaration and definition”, Syntax Rule 2) rewrites the <graph initializer> not containing a <graph expression> to the form containing a <graph expression>. Although Subclause 10.4, “Value variable and parameter declaration and definition” has a similar format with the <value initializer>, it does not specify a similar rewrite. This should be revisited and harmonized.

Solution:

Editor's Notes for IWD 39075:202y(E)
Possible Problems

None provided with comment.

GQL-255 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 18.3, “<comparison predicate>”.

Note At: Editor's Note number 218.

Source: W21-058.

Possible Problem:

Comparison of numeric value types having different base types should be further reviewed.

Solution:

None provided with comment.

GQL-257 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 12.2, “<call procedure statement>”.

Note At: Editor's Note number 72.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.

Subclause 12.2, “<call procedure statement>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-259 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 14.2, “<do statement>”.

Note At: Editor's Note number 88.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.

Subclause 14.2, “<do statement>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-260 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 14.3, “<insert statement>”.

Note At: Editor's Note number 91.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
Subclause 14.3, "<insert statement>" does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-262 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 14.4, "<set statement>".

Note At: Editor's Note number 94.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
Subclause 14.4, "<set statement>" does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-263 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 14.5, "<remove statement>".

Note At: Editor's Note number 97.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
Subclause 14.5, "<remove statement>" does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-264 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 14.6, "<delete statement>".

Note At: Editor's Note number 100.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
Subclause 14.6, "<delete statement>" does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

Editor's Notes for IWD 39075:202y(E)
Possible Problems

GQL-266 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 15.2, “<composite query expression>”.

Note At: Editor's Note number 105.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.

Subclause 15.2, “<composite query expression>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-267 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 15.5.1, “<match statement>”.

Note At: Editor's Note number 107.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.

Subclause 15.5.1, “<match statement>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-268 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 15.6.5, “<aggregate statement>”.

Note At: Editor's Note number 118.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.

Subclause 15.6.5, “<aggregate statement>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-269 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 15.6.6, “<for statement>”.

Note At: Editor's Note number 122.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
Subclause 15.6.6, "<for statement>" does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-270 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 15.7.2, "<return statement>".

Note At: Editor's Note number 128.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
Subclause 15.7.2, "<return statement>" does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-271 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 16.14, "<inline procedure call>".

Note At: Editor's Note number 172.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
Subclause 16.14, "<inline procedure call>" does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-272 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 16.15, "<named procedure call>".

Note At: Editor's Note number 178.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
Subclause 16.15, "<named procedure call>" does not do that yet and requires corresponding Syntax Rules.

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Solution:

None provided with comment.

GQL-273 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 16.17, “<group by clause>”.

Note At: Editor's Note number 186.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
Subclause 16.17, “<group by clause>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-274 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 16.19, “<aggregate function>”.

Note At: Editor's Note number 191.

Source: BER-019

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
The Subclause does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-275 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 16.20, “<sort specification list>”.

Note At: Editor's Note number 194

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result.
Subclause 16.20, “<sort specification list>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-277 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 17.4, "Binding table references".

Note At: Editor's Note number 203.

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result. Subclause 17.4, "Binding table references" does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-278 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 19.20, "<value query expression>".

Note At: Editor's Note number 251

Source: BER-019.

Possible Problem:

BER-019 has established declared type propagation for working record, working table, and result. Subclause 19.20, "<value query expression>" does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-280 The following Possible Problem has been noted:

Severity: minor technical

Reference:

« WG3:BER-031 »

No specific location.

Note At: None.

Source: WG3:BER-100.

Possible Problem:

The notion of a "most specific type" seems out-of-date by [W12-029] and [BER-040r3] and needs to be removed consistently from the document.

Solution:

None provided with comment.

GQL-284 The following Possible Problem has been noted:

Severity: minor technical

Reference:

« WG3:BER-040R3 »

No specific location.

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Note At: None.

Source: WG3:BER-040R3.

Possible Problem:

The rules for store assignment must be defined.

Solution:

None provided with comment.

GQL-289 The following Possible Problem has been noted:

Severity: minor technical

Reference:

[« WG3:BER-094R1 »](#)

Subclause 19.4, “<numeric value expression>”.

Note At: Editor's Note number 226.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established declared type propagation for the result. Subclause 19.4, “<numeric value expression>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-290 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 19.6, “<numeric value function>”.

Note At: Editor's Note number 233.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established declared type propagation for the result. Subclause 19.6, “<numeric value function>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-291 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 19.8, “<string value function>”.

Note At: Editor's Note number 237.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established declared type propagation for the result. Subclause 19.8, “<string value function>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-292 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 19.9, “<datetime value expression>”.

Note At: Editor's Note number 238.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established declared type propagation for the result. Subclause 19.9, “<datetime value expression>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-293 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 19.10, “<datetime value function>”.

Note At: Editor's Note number 240.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established declared type propagation for the result. Subclause 19.10, “<datetime value function>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-294 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 19.11, “<duration value expression>”.

Note At: Editor's Note number 242.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established declared type propagation for the result. Subclause 19.11, “<duration value expression>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-295 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 19.12, “<duration value function>”.

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Note At: Editor's Note number 244.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established declared type propagation for the result. Subclause 19.12, “<duration value function>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-296 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 19.16, “<map value constructor>”.

Note At: Editor's Note number 246.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established declared type propagation for the result. Subclause 19.16, “<map value constructor>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-297 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 19.19, “<property reference>”.

Note At: Editor's Note number 248.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established declared type propagation for the result. Subclause 19.19, “<property reference>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-298 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 19.20, “<value query expression>”.

Note At: Editor's Note number 251.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established declared type propagation for the result. Subclause 19.20, “<value query expression>” does not do that yet and requires corresponding Syntax Rules.

Solution:

None provided with comment.

GQL-299 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 21.11, "Result of value type combinations".

Note At: Editor's Note number 279.

Source: WG3:BER-094R1.

Possible Problem:

[BER-019] and [BER-094R1] established that rules on types should be specified in Syntax Rules. Subclause 21.11, "Result of value type combinations" has to be adjusted accordingly, as well as the wording of every rule that calls the subclause.

Solution:

None provided with comment.

GQL-300 The following Possible Problem has been noted:

Severity: minor technical

Reference: Subclause 13.6, "<drop graph statement>", Subclause 13.14, "<drop graph type statement>".

Note At: Editor's Note number 79, Editor's Note number 82.

Source: Editor.

Possible Problem:

It is not clear how <graph name> (<graph type name>) can be immediately contained in <catalog graph parent and name> (<catalog graph type parent and name>) immediately contains <curl path parameter> and not <graph parent specification> <graph name> (<graph type name>). It is also not clear if or how it is confirms that that name identifies the correct type of object.

Solution:

None provided with comment.

Possible Problems: Major Editorial

[GQL-098] The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 4.12, "Data types".

Note At: Editor's Note number 29, Editor's Note number 42, Editor's Note number 44, Editor's Note number 49, Editor's Note number 261.

Source: W05-020.

Possible Problem:

W05-020 suggests that, whilst the introductory text is based on that in SQL/Foundation, the Subclause as a whole has not yet reached consensus agreement.

W12-029 agreed on the following Subclauses:

- Subclause 4.12.1, "General introduction to data types".
- Subclause 4.12.2, "Naming of predefined value types and associated base types".
- Subclause 4.12.3, "Data type descriptors".

Solution:

None provided with comment.

[GQL-099] The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 4.16.3, "Predefined value types".

Note At: Editor's Note number 52, Editor's Note number 55.

Source: W05-020.

Possible Problem:

W05-020 suggests that, whilst this Subclause is not considered controversial and has already been discussed in [W03-034], there is still a need to establish consensus.

W12-029 agreed on the following subclauses:

- Subclause 4.16.3.1, "Boolean types".
- Subclause 4.16.3.2, "Character string types".
- Subclause 4.16.3.3, "Byte string types".
- Subclause 4.16.3.4, "Numeric types".

Solution:

None provided with comment.

[GQL-107] The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 15.5.2, "<call query statement>".

Note At: Editor's Note number 108.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [BNE-023], needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-108 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 15.6.1, "<mandatory statement>".

Note At: Editor's Note number 111.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [BNE-023], needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-109 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 15.6.2, "<optional statement>".

Note At: Editor's Note number 113.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [BNE-023], needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-111 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 15.6.4, "<let statement>".

Note At: Editor's Note number 115.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [BNE-023] as WITH, needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-112 The following Possible Problem has been noted:

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Severity: major editorial

Reference: Subclause 15.6.5, “<aggregate statement>”.

Note At: Editor's Note number 116.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [BNE-023] as WITH, needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-113 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 15.6.6, “<for statement>”.

Note At: Editor's Note number 119, Editor's Note number 120, Editor's Note number 121.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [BNE-023] as UNWIND, needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-116 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 15.7.2, “<return statement>”.

Note At: Editor's Note number 125.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [JCJ-010r1] and [BNE-023], needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-117 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 15.7.3, “<select statement>”.

Note At: Editor's Note number 131, Editor's Note number 134.

Source: W05-020, W15-017.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [JCJ-010r1], needs further discussion to establish consensus. Furthermore, the Syntax and General Rules are probably incomplete and, in any case, need to be reviewed.

Solution:

None provided with comment.

GQL-118 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 16.13, "<procedure call>".

Note At: Editor's Note number 169.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [BNE-023], needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-119 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 16.14, "<inline procedure call>".

Note At: Editor's Note number 171, Editor's Note number 174.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [BNE-023], needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-120 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 16.15, "<named procedure call>".

Note At: Editor's Note number 175, Editor's Note number 179, Editor's Note number 180, Editor's Note number 181, Editor's Note number 182.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [BNE-023], needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-123 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 16.19, "<aggregate function>".

Note At: Editor's Note number 187.

Editor's Notes for IWD 39075:202y(E)
Possible Problems

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was discussed in [BNE-023], needs further discussion to establish consensus.

Solution:

None provided with comment.

GQL-127 The following Possible Problem has been noted:

Severity: major editorial

Reference: Clause 17, "Object references".

Note At: Editor's Note number 197.

Source: W05-020.

Possible Problem:

W05-020 suggests that this clause, which is based on [MMX-055] and discussed in [W03-034], needs further discussion to establish consensus.

W13-024 agreed on Subclause 17.1, "Schema references".

All primary object references should be resolved by Syntax Rules.

<schema reference>s are resolved by Syntax Rules.

Solution:

None provided with comment.

GQL-228 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 15.2, "<composite query expression>".

Note At: Editor's Note number 106.

Source: W15-017.

Possible Problem:

How the columns of the two tables are related and what the result of data type combinations is, need to be specified.

Solution:

None provided with comment.

GQL-239 The following Possible Problem has been noted:

Severity: major editorial

Reference: Subclause 4.11, "Graph pattern matching".

Note At: Editor's Note number 16.

Source: W16-034R2.

Possible Problem:

The WG3 W16 Meeting accepted the text of W16-034R2 for application also in GQL. However, it was recognised that some enhancement would probably be necessary to cope with the enhanced facilities of GQL.

Solution:

None provided with comment.

Possible Problems: Minor Editorial

[GQL-135] The following Possible Problem has been noted:

Severity: minor editorial

Reference: Clause 1, "Scope".

Note At: Editor's Note number 1.

Source: W05-020.

Possible Problem:

W05-020 suggests that this clause should be reviewed for adherence to the NWIP scope, adherence to current ISO directives, completeness, and readability.

Solution:

None provided with comment.

[GQL-136] The following Possible Problem has been noted:

Severity: minor editorial

Reference: Clause 3, "Terms and definitions".

Note At: Editor's Note number 2.

Source: SXM-052, W05-020, W09-016, W09-021R1, W09-036, W10-019R1, W11-031, W12-019, W12-029, W13-010R1, W19-017.

Possible Problem:

W05-020 suggests that, although much of this clause taken from SQL/PGQ and SQL Foundation, and is not expected to be controversial, it should nevertheless be reviewed.

The following terms from Subclause 3.2, "General terms and definitions", require further review of their associated definitions and regarding their general inclusion in the document:

1) variable <"x variable"

The following terms from Subclause 3.3, "Graph terms and definitions", require further review of their associated definitions and regarding their general inclusion in the document:

1) empty graph

2) directed edge

« WG3:BER-081 deleted one item »

The following terms from Subclause 3.4, "GQL-environment terms and definitions", require further review of their associated definitions and regarding their general inclusion in the document:

1) GQL-environment

2) GQL-server

3) principal

4) GQL-client

5) GQL-agent

« WG3:BER-040R3 deleted one item »

6) session-derived <GQL-object>

7) GQL-transaction

« WG3:BER-040R3 deleted nine items »

8) original

9) derived

The following terms from Subclause 3.5, "GQL-catalog terms and definitions", require further review of their associated definitions and regarding their general inclusion in the document:

1) library

« WG3:BER-082R1 deleted one item »

2) type object

« WG3:BER-040R3 deleted one item »

3) catalog-derived <object>

4) visible <GQL-object>

5) home schema

6) home graph

The following terms from Subclause 3.6, "Procedure terms and definitions", require further review of their associated definitions and regarding their general inclusion in the document:

1) side effect

2) catalog-modifying <side effect>

3) session-modifying <side effect>

4) transaction-modifying <side effect>

5) data-populating <side effect>

6) data-modifying <side effect>

« WG3:BER-060 deleted one item »

7) result type

« WG3:BER-081 deleted three items »

8) locally-derived <object>

9) catalog-modifying procedure

10) data-modifying procedure

11) stored procedure

12) procedure

13) procedure signature <procedure>

14) procedure logic <procedure>

15) formal parameter

16) parameter cardinality <formal parameter>

Editor's Notes for IWD 39075:202y(E)
Possible Problems

- 17) single parameter cardinality
- 18) multiple parameter cardinality
- 19) simple view
- 20) parameterized view
- 21) view
- 22) query
- 23) function
- 24) command
- 25) session command
- 26) transaction command
- 27) GQL-procedure
- 28) external procedure
- 29) statement

« WG3:BER-081 deleted three items »

The following terms from Subclause 3.7, “Procedure syntax terms and definitions”, require further review of their associated definitions and regarding their general inclusion in the document:

- 1) pattern
- 2) pattern macro
- 3) path pattern macro
- 4) pattern variable <“x pattern variable”
- 5) pattern match
- 6) element pattern
- 7) label expression
- 8) linear composition
- 9) whitespace

The following terms from Subclause 3.8, “Value terms and definitions”, require further review of their associated definitions and regarding their general inclusion in the document:

« WG3:BER-040R3 deleted two items »

- 1) character string

The following terms from Subclause 3.9, “Type terms and definitions”, require further review of their associated definitions and regarding their general inclusion in the document:

- 1) any type
- 2) graph type
- 3) nothing type

Solution:

The following terms are directly inherited from SQL with marginal editorial modifications. All of these terms may still be subject to further review as may be indicated by additional Possible Problems or Language Opportunities.

- 1) scope (of a name or declaration)

In the following we cite change proposals that have agreed on the inclusion of some terms and their associated definition in this document. All of these terms may still be subject to further review as may be indicated by additional Possible Problems or Language Opportunities.

BER-019 agreed on the terms:

- 1) Variable
- 2) Binding variable
- 3) Fixed variable
- 4) Iterated variable
- 5) Value variable
- 6) graph variable
- 7) Binding table variable
- 8) Binding table type
- 9) Declared type

W19-017R2 agreed on the terms:

- 1) scope (of a working object)
- 2) scope clause
- 3) declaration <"x declaration"
- 4) definition <"x definition"

W13-010R1 agreed on the terms:

- 1) descriptor
- 2) persistent
- 3) dictionary
- 4) temporary
- 5) GQL-catalog
- 6) GQL-catalog-root
- 7) GQL-directory
- 8) GQL-schema
- 9) catalog object
- 10) GQL-data
- 11) GQL-object
- 12) data object

Editor's Notes for IWD 39075:202y(E)
Possible Problems

- 13) primary object
- 14) secondary object

W12-029 agreed on the terms:

- 1) atomic
- 2) composite
- 3) identify
- 4) identifier
- 5) object <"X" object>
- 6) multiset
- 7) set
- 8) sequence
- 9) ordered set
- 10) instance <"X" instance>
- 11) site
- 12) reference <"X" reference>
- 13) literal
- 14) referent <"X" referent>
- 15) data type
- 16) supertype <"X" supertype>
- 17) subtype <"X" subtype>
- 18) meta type
- 19) base type
- 20) most specific type <of "X">
- 21) constructed <data type>
- 22) user-defined <data type>
- 23) predefined <data type>
- 24) atomic <data type>
- 25) nullable <data type>
- 26) object type
- 27) value type
- 28) reference value type
- 29) property value type
- 30) value <"X" value>

- 31) direct <value>
- 32) indirect <value>
- 33) reference value
- 34) material <value>
- 35) null value

W12-019 agreed on the terms:

- 1) dictionary
- 2) session context
- 3) current session context
- 4) session parameter
- 5) session parameter flag
- 6) GQL-request
- 7) request source
- 8) request parameter
- 9) GQL-request context
- 10) current request context
- 11) execution stack
- 12) context parameter
- 13) side effect
- 14) execution
- 15) operation
- 16) execution context
- 17) current execution context
- 18) execution outcome
- 19) current execution outcome
- 20) successful outcome
- 21) failed outcome
- 22) regular result
- 23) omitted result
- 24) current execution result
- 25) evaluation
- 26) working schema
- 27) working graph

Editor's Notes for IWD 39075:202y(E)
Possible Problems

- 28) working table
- 29) working record
- 30) material
- 31) null value

W14-014 deleted the term:

- 1) current

W10-019R1 agreed on the terms:

- 1) edge variable
- 2) element (graph element)
- 3) endpoint
- 4) graph pattern
- 5) label
- 6) path
- 7) path pattern
- 8) property
- 9) graph (property graph)
- 10) subpath
- 11) undirected edge
- 12) node (vertex)
- 13) node variable

W09-036 agreed on the terms:

- 1) multigraph
- 2) mixed graph
- 3) directed graph
- 4) undirected graph
- 5) directionality <edge>
- 6) undirected edge
- 7) any-directed edge
- 8) endpoint <edge>
- 9) endpoint <path>
- 10) source node (start node)
- 11) destination node (end node)

W09-021R1 agreed on the terms:

- 1) binding table
- 2) record
- 3) field
- 4) column name
- 5) column type
- 6) column
- 7) record type
- 8) field type

W09-016 agreed on the terms:

- 1) node variable
- 2) node pattern
- 3) edge pattern

SXM-052 agreed on the terms:

- 1) graph pattern variable
- 2) element variable
- 3) node (vertex) variable
- 4) edge (relationship) variable
- 5) path variable
- 6) subpath variable

« WG3:BER-040R3 »

BER-040R3 agreed on the terms:

- 1) GQL-session
- 2) static object type
- 3) static object
- 4) static site
- 5) static
- 6) dynamic object type
- 7) dynamic object
- 8) dynamic site
- 9) dynamic
- 10) boxed value
- 11) constant object
- 12) property value
- 13) property type <node type or edge type definition>

Editor's Notes for IWD 39075:202y(E)
Possible Problems

« WG3:BER-060 »

BER-060 agreed on the terms:

- 1) supported result

« WG3:BER-081 »

BER-081 agreed on the terms:

- 1) adjacent
- 2) result object
- 3) result object type
- 4) locally-defined <object>
- 5) successful result statement
- 6) successful statement
- 7) failed statement

« WG3:BER-082R1 »

BER-082R1 agreed on the terms:

- 1) alias

GQL-141 The following Possible Problem has been noted:

Severity: minor editorial

Reference: Subclause 4.3.5, "Binding tables".

Note At: Editor's Note number 9.

Source: W05-020, W09-021R1.

Possible Problem:

W05-020 suggests that this Subclause, which was accepted and adapted from [MMX-055], should be reviewed.

W09-021R1 agreed on Subclause 4.3.5, "Binding tables", except for:

- The paragraph starting "The collection value for a binding table is defined ..." and item list following it.

Solution:

None provided with comment.

GQL-142 The following Possible Problem has been noted:

Severity: minor editorial

Reference: Subclause 4.3.6, "Libraries".

Note At: Editor's Note number 10.

Source: W05-020.

Possible Problem:

W05-020 suggests that this Subclause, which was accepted and adapted from [MMX-055], should be reviewed.

Solution:

None provided with comment.

GQL-147 The following Possible Problem has been noted:

Severity: minor editorial

Reference: Clause 13, "Catalog-modifying statements".

Note At: Editor's Note number 73.

Source: W05-020.

Possible Problem:

W05-020 suggests that this clause, which was accepted and adapted from [MMX-055] and [MMX-028r2], should be reviewed.

Solution:

None provided with comment.

GQL-188 The following Possible Problem has been noted:

Severity: minor editorial

Reference: Subclause 21.9, "Ordering operations".

Note At: Editor's Note number 275.

Source: Editors.

Possible Problem:

The list of ordering operations needs to be fully specified.

Solution:

None provided with comment.

GQL-256 The following Possible Problem has been noted:

Severity: minor editorial

Reference:

« Email from: Jan Michels 2022-06-16 »

Subclause 5.3.4, "Descriptors".

Note At: Editor's Note number 58, Editor's Note number 59.

Source: Email from Jan Michels 2022-06-16.

Possible Problem:

The definitions of "generally include" and "generally depend" are currently not used in the document.
If they are still not used by the time the document exists the DIS stage, they should be removed.

Solution:

None provided with comment.

Language Opportunities

GQL-003 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 4.10.2, "Procedures", Subclause 16.15, "<named procedure call>".

Note At: Editor's Note number 12, Editor's Note number 177.

Source: Editors.

Language Opportunity:

Bindings for host languages should be defined. This should include calling GQL procedures from other languages, e.g., Java, C++, Python, and calling functions and/or procedure, which are written in another language, from within GQL.

Solution:

None provided with comment.

GQL-004 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Clause 13, "Catalog-modifying statements".

Note At: Clause 13, "Catalog-modifying statements", Editor's Note number 74.

Source: Editors.

Language Opportunity:

The GQL schema and meta-graph need to be defined together with the statements to manipulate it.

Solution:

None provided with comment.

GQL-005 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference:

« WG3:BER-040R3 »

All Access Rule sections.

Note At: None.

Source: Editors, W12-012, BER-040R3.

Language Opportunity:

The Security Model for GQL and related concepts such as roles need to be defined.

Solution:

None provided with comment.

GQL-011 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Clause 20, "Lexical elements".

Note At: Editor's Note number 263, Editor's Note number 225, Editor's Note number 61.

Source: Editors.

Language Opportunity:

SQL pays a lot of attention as to whether an expression is deterministic or not, as this has consequences for its use in constraints. If GQL is to have a schema for graphs that includes constraints, then this will also need considerable attention.

Solution:

None provided with comment.

GQL-012 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 18.3, “<comparison predicate>”, Subclause 21.8, “Equality operations”, Subclause 21.9, “Ordering operations”, Annex F, “Differences with SQL”.

Note At: Editor's Note number 220, Editor's Note number 274, Editor's Note number 276, Editor's Note number 281.

Source: Editors, WG3:W21-058.

Language Opportunity:

Consider explicit support for additional collations other than UCS_BASIC and UNICODE (e.g., case insensitive variants, use of alternative collation tables).

Solution:

None provided with comment.

GQL-017 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 15.7.2, “<return statement>”, Subclause 15.7.3, “<select statement>”, Subclause 16.17, “<group by clause>”, Subclause 16.19, “<aggregate function>”, Subclause 19.5, “<value expression primary>”.

Note At: Editor's Note number 188, Editor's Note number 127, Editor's Note number 133, Editor's Note number 185, Editor's Note number 185, Editor's Note number 193, Editor's Note number 227.

Source: Editors.

Language Opportunity:

Aggregation would benefit from being improved for the needs of GQL:

- Revisit the set of provided aggregation functions in light of existing products and market needs (e.g., add TIMES or graph-specific aggregation functions).
- Provide means to better control how aggregation functions behave on empty inputs (no rows), e.g., by allowing the user to specify a default value or default failure behavior for aggregation on empty inputs.
- Allow aggregation to scope over whole subqueries (partition by), possibly taking advantage of using stable variables as implicit grouping keys and supporting the evaluation of aggregation functions over the incoming working table in every statement.

See also Language Opportunity **GQL-186**.

Solution:

None provided with comment.

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

GQL-023 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.15, “<named procedure call>”.

Note At: Editor's Note number 176.

Source: Editors.

Language Opportunity:

Decide on support for passing procedures, queries, functions as parameters.

Solution:

None provided with comment.

GQL-025 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.7, “<path pattern expression>”.

Note At: Editor's Note number 160.

Source: SXM-052.

Language Opportunity:

Path pattern union is not defined using left recursion. SXM-052 believed that it should be possible to support left recursion but declined to do so for expediency. It is a Language Opportunity to support left recursion.

Linked to Language Opportunity **PGQ-019**.

Solution:

None provided with comment.

GQL-029 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 20.6, “<token> and <separator>”.

Note At: Editor's Note number 262.

Source: Editors.

Language Opportunity:

It is a Language Opportunity to extend the definition of <regular identifier>s to use the Unicode XID_Start and XID_Continue instead of the more restricted ID_Start and ID_Continue.

Solution:

None provided with comment.

GQL-030 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 4.6.2, “Transaction demarcation”, and others.

Note At: Editor's Note number 11.

Source: Editors.

Language Opportunity:

Currently transaction demarcation is defined such that any failure within a GQL-request procedure will cause a rollback attempt. Rollback on failure may not be the best option here. Consider an application that has multiple procedure invocations within a transaction context where the first N procedures succeed but procedure n+1 fails. It would be preferable to decide in the application logic whether to commit or rollback, e.g., by using a session level configuration mechanism or by providing options to transaction demarcation commands.

Solution:

None provided with comment.

GQL-032 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 19.8, “<string value function>”.

Note At: Editor's Note number 234, Editor's Note number 236, Editor's Note number 235.

Source: Editors.

Language Opportunity:

SQL, Cypher, and possibly other property graph query languages have various styles of regular expression substring functions as well as other possibly relevant string value functions that GQL currently is missing. It is a Language Opportunity to add functionality equivalent to at least that of SQL to GQL.

Solution:

None provided with comment.

GQL-034 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.11, “<simplified path pattern expression>”.

Note At: Editor's Note number 168.

Source: MMX-060.

Language Opportunity:

It has been proposed that a macro name may be a <simplified primary> in a <simplified path pattern expression>.

Linked to Language Opportunity **PGQ-035**.

Solution:

None provided with comment.

GQL-035 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference:

« **WG3:BER-050** »

Subclause 14.3, “<insert statement>”.

Note At: Editor's Note number 89.

Source: MMX-047.

Language Opportunity:

Editor's Notes for IWD 39075:202y(E)

Language Opportunities

Discussion paper MMX-047 suggests the addition of a “Time To Live” option, which would require specified graph elements be deleted after a certain time to save storage space.

Solution:

None provided with comment.

GQL-036 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.7, “<path pattern expression>”.

Note At: Editor's Note number 159 (Syntax Rule 8)).

Source: W01-014.

Language Opportunity:

It may be possible to permit nested quantifiers. WG3:W01-014 contained a discussion of a way to support aggregates at different depths of aggregation if there are nested quantifiers.

Linked to Language Opportunity **PGQ-036**.

Solution:

None provided with comment.

GQL-044 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 21.3, “Evaluation of a <path pattern expression>”.

Note At: Editor's Note number 268.

Source: W04-009R1.

Language Opportunity:

It may be possible to enforce implicit joins of unconditional singletons exposed by a <path concatenation> as part of the GRs for <path concatenation>. This was discussed in a SQL/PG ad hoc meeting on September 8, 2020. It was decided not to attempt that change as part of WG3:W04-009R1, leaving it as a future possibility.

Linked to Language Opportunity **PGQ-047**.

Solution:

None provided with comment.

GQL-051 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.7, “<path pattern expression>”.

Note At: Editor's Note number 165.

Source: W04-009R1.

Language Opportunity:

WG3:W04-009R1 recognized that a graph query may have a sequence of MATCH clauses, with the bindings of one MATCH clause *MC1* visible in all subsequent MATCH clauses in the same invocation of <graph table>, and that it should be permissible to reference such variables in any <parenthesized path pattern where clause> simply contained in a subsequent MATCH clause *MC2*. The relevance of this LO to GQL needs to be investigated.

Linked to Language Opportunity [PGQ-051](#).

Solution:

None provided with comment.

GQL-052 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.6, “<path pattern prefix>”.

Note At: Editor's Note number 152.

Source: W04-009R1.

Language Opportunity:

The ability to specify “cheapest” queries (analogous to SHORTEST but minimizing the sum of costs along a path) is desirable.

Linked to Language Opportunity [PGQ-052](#).

Solution:

None provided with comment.

GQL-053 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.6, “<path pattern prefix>”.

Note At: Editor's Note number 155.

Source: W04-009R1.

Language Opportunity:

In addition to SHORTEST GROUP, it has been proposed to support SHORTEST [k] WITH TIES, with the semantics to return the first k matches (where k defaults to 1) when sorting matches in ascending order on number of edges, and also return any matches that have the same number of edges as the last of the k matches. This is the semantics of WITH TIES in Subclause 7.17, “<query expression>” in SQL/Foundation.

Linked to Language Opportunity [PGQ-053](#).

Solution:

None provided with comment.

GQL-054 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: No specific location.

Source: W04-009R1, W09-031.

Language Opportunity:

WG3:W04-009R1 believed it should be possible to support nested selectors using recursion but did not undertake that. It is a Language Opportunity to support nested selectors.

Linked to Language Opportunity [PGQ-054](#).

Solution:

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

None provided with comment.

GQL-055 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: No specific location.

Source: W04-009R1, W09-031.

Language Opportunity:

WG3:W04-009R1 believed it should be possible to support selectors within quantifiers with a finite upper bound, but did not undertake that. (This comment applies to a static upper bound, not dynamic, since we are forbidding multiple selectors within a restrictor.) It is a Language Opportunity to support selectors within quantifiers with a finite upper bound.

Linked to Language Opportunity **PGQ-055**.

Solution:

None provided with comment.

GQL-056 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.6, “<path pattern prefix>”.

Note At: Editor's Note number 156.

Source: W04-009R1.

Language Opportunity:

With more work, it is possible to recognize when a node variable is declared uniformly in the first or the last position in every operand of a <path pattern union>. However, WG3:W04-009R1 declined to make the effort because it is easy for the user to factor out such a node pattern. For example, instead of

(X) -> (Y) | (X) -> (Z)

the user can write

(X) (-> (Y) | -> (Z))

Thus, a more general definition of right or left boundary variable is possible.

Linked to Language Opportunity **PGQ-056**.

Solution:

None provided with comment.

GQL-057 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.5, “<graph pattern>”.

Note At: Editor's Note number 142.

Source: W04-009R1, W09-031.

Language Opportunity:

It has been suggested that it might be possible to treat the <path pattern prefix> specified in <keep clause> as merely providing a default <path pattern prefix> rather than a mandatory one for each <path pattern>. Whereas nested <path pattern prefix> is prohibited, this may be a feasible avenue of growth. On the other hand, perhaps a less definitive verb than KEEP may be appropriate when specifying a default <path pattern prefix>.

Linked to Language Opportunity [PGQ-049](#).

Solution:

None provided with comment.

GQL-059 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.5, "<graph pattern>".

Note At: Editor's Note number 147.

Source: W04-009R1.

Language Opportunity:

It has been proposed that unconditional singletons exposed by prior MATCH clauses may also be joined implicitly.

Linked to Language Opportunity [PGQ-050](#).

Solution:

None provided with comment.

GQL-161 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.21, "<limit clause>".

Note At: Editor's Note number 195.

Source: Editors.

Language Opportunity:

Additional, commonly supported subclauses of <limit clause> should be added.

Solution:

None provided with comment.

GQL-162 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.22, "<offset clause>".

Note At: Editor's Note number 196.

Source: Editors.

Language Opportunity:

Additional, commonly supported subclauses of <offset clause> should be added.

Solution:

None provided with comment.

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

GQL-163 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 15.6.7, “`<order by and page statement>`”.

Note At: Editor's Note number 123.

Source: Editors, W15-020.

Language Opportunity:

Additional support for PARTITION BY, WITH TIES, WITH INDEX, and WITH ORDINALITY should be considered.

Solution:

None provided with comment.

GQL-168 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 12.2, “`<call procedure statement>`”.

Note At: Editor's Note number 71.

Source: Editors.

Language Opportunity:

Stand-alone calls need to be added (standalone calls are short-hand syntax for just calling a single GQL-procedure).

Solution:

None provided with comment.

GQL-169 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 12.2, “`<call procedure statement>`”.

« WG3:BER-050 »

Note At: Editor's Note number 70, Editor's Note number 87, Editor's Note number 90, Editor's Note number 93, Editor's Note number 96, Editor's Note number 99, Editor's Note number 112, Editor's Note number 114, Editor's Note number 126, Editor's Note number 136.

Source: Editors.

Language Opportunity:

The addition of optional `<where clause>`s should be considered throughout the document.

Solution:

None provided with comment.

GQL-171 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Annex F, “Differences with SQL”.

Note At: Editor's Note number 282.

Source: Editors.

Language Opportunity:

Support for large character string objects needs to be specified.

Solution:

None provided with comment.

GQL-175 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 19.2, “<value expression>”, Subclause 19.14, “<list value function>”.

« WG3:BER-094R1 »

Note At: None.

Source: Editors.

Language Opportunity:

It needs to be decided which additional value functions on collections and graph elements should be included.

Solution:

None provided with comment.

GQL-176 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 19.6, “<numeric value function>”.

Note At: Editor's Note number 228, Editor's Note number 229, Editor's Note number 230, Editor's Note number 231, Editor's Note number 232.

Source: Editors.

Language Opportunity:

It needs to be decided which additional numeric value functions should be included.

Solution:

None provided with comment.

GQL-177 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 18.2, “<predicate>”.

Note At: Editor's Note number 215, Editor's Note number 216, Editor's Note number 217.

Source: Editors.

Language Opportunity:

It needs to be decided which additional predicates should be included.

Solution:

None provided with comment.

GQL-181 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: No specific location.

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

Source: W08-018, W09-031.

Language Opportunity:

W08-018 prohibited a selector contained in a <path pattern union> or <path multiset alternation> *PU*, but believed that if *PU* is at the “top” of a path pattern, and a selector is at the “top” of an operand of *PU*, then this scenario does not violate compositionality. It is a Language Opportunity to permit this scenario.

Linked to Language Opportunity **GQ-060**.

Solution:

None provided with comment.

GQL-185 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 4.16.3.5, “Temporal types”.

Note At: Editor's Note number 56.

Source: Editors.

Language Opportunity:

Support for system-versioned graphs should be added.

Solution:

None provided with comment.

GQL-186 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.13, “<procedure call>”, Subclause 16.19, “<aggregate function>”.

Note At: Editor's Note number 170, Editor's Note number 190.

Source: Editors.

Language Opportunity:

Support for grouped procedure calls needs to be specified. A grouped procedure call invokes a procedure for each partition of the binding table and combines (concatenates) all results received. Partitions may be determined by a user-specified grouping key or perhaps sets of grouping keys.

See also Language Opportunity **GQL-017**.

Solution:

A possible solution might all a syntactic form like:

```
CALL proc(args) PER key YIELD results
```

or perhaps rely on per-query block partitioning:

```
PER key
...
CALL proc(args) YILED results
...
```

GQL-194 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.5, “<graph pattern>”, Subclause 21.2, “Machinery for graph pattern matching”.

Note At: Editor's Note number 143, Editor's Note number 266.

Source: Editors.

Language Opportunity:

Projection and handling of subpath variables needs to be specified.

Solution:

None provided with comment.

GQL-196 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 19.10, “<datetime value function>”.

Note At: Editor's Note number 239.

Source: Editors.

Language Opportunity:

Cypher contains additional datetime functions. It is a Language Opportunity to add equivalent functionality to GQL.

Solution:

None provided with comment.

GQL-197 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 19.12, “<duration value function>”.

Note At: Editor's Note number 243.

Source: Editors.

Language Opportunity:

Cypher contains additional duration functions. It is a Language Opportunity to add equivalent functionality to GQL.

Solution:

None provided with comment.

GQL-199 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 15.6.5, “<aggregate statement>”.

Note At: Editor's Note number 117.

Source: Editors.

Language Opportunity:

Consider extending aggregate with GROUP BY.

Solution:

None provided with comment.

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

GQL-212 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.7, "<path pattern expression>".

Note At: Editor's Note number 157.

Source: W09-017, W09-036.

Language Opportunity:

In the BNF for <full edge any direction>, the delimiter tokens <~[]~> have been suggested as a synonym for -[]- as part of Feature GA10, "Undirected edge patterns". The synonym for the <abbreviated edge pattern> - (<minus sign>) would then be <~>, the synonym for <simplified defaulting any direction> would use the delimiter tokens <~/ /~> and the synonym for <simplified override any direction> would use the tokens <~ and > surrounding a label as originally proposed in MMX-060. These synonyms might be considered to make the table of edge patterns more harmonious and internally consistent.

Linked to Language Opportunity **PGQ-062**.

Solution:

None provided with comment.

GQL-213 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 18.2, "<predicate>".

Note At: None.

Source: W10-017.

Language Opportunity:

SQL/Foundation defines five operators that use XQuery regular expression syntax:

```
LIKE_REGEX  
OCCURRENCES_REGEX  
POSITION_REGEX  
SUBSTRING_REGEX  
TRANSLATE_REGEX
```

These REGEX operators could be very useful and should be considered.

Solution:

None provided with comment.

GQL-217 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 4.16.3.4, "Numeric types".

Note At: Editor's Note number 53.

Source: W12-029.

Language Opportunity:

Support for approximate numeric types that are compatible with the arithmetic formats for which ISO/IEC 60559:2020/IEEE 754:2019 defines interchange formats should be added. This needs to

include provisions for infinity values and literals, NaN values and literals, rounding, casting, error handling, data types, the modification of existing operations, and related conformance features.

Solution:

None provided but see paper W11-015 for a discussion.

GQL-218 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 4.16.3.4, "Numeric types".

Note At: None.

Source: W12-029.

Language Opportunity:

In SQL, it is implementation-defined whether the loss of non-zero bytes due to truncation raises an exception or not. This should be reconsidered by GQL.

Solution:

None provided with comment.

GQL-219 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 3.8, "Value terms and definitions".

Note At: Editor's Note number 4.

Source: W12-019.

Language Opportunity:

In a system implementing both SQL and GQL, how would GQL consume a SQL table? Besides providing syntax, a solution to this LO has to answer the question how GQL should support mapping SQL tables with multiple columns with the same name into GQL?

Solution:

None provided with comment.

GQL-221 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 19.22, "<cast specification>".

Note At: None.

Source: W13-012

Language Opportunity:

Many programming languages support converting boolean types to numerics where *True* converts to 1 (one) and *False* converts to 0 (zero). Such conversions could be useful to GQL users.

Solution:

None provided with comment.

GQL-222 The following Language Opportunity has been noted:

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

Severity: Language Opportunity

Reference: Subclause 19.22, “<cast specification>”.

Note At: None.

Source: W13-012

Language Opportunity:

Many databases and programming languages include individual type conversion functions such as:

- toBoolean
- toInteger
- toFloat
- toString

If there is a need at some point to add individual conversion functions to GQL, they could be specified as syntactic transformations to the appropriate CAST function.

Solution:

None provided with comment.

GQL-223 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 19.22, “<cast specification>”.

Note At: None.

Source: W13-012

Language Opportunity:

In ISO/IEC 9075-2:202x, <cast specification> includes an optional FORMAT <cast template> that allows a user to provide a format when converting a datetime to a character string. A similar capability could be useful in GQL.

Solution:

None provided with comment.

GQL-224 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 19.22, “<cast specification>”.

Note At: None.

Source: W13-012

Language Opportunity:

In a CAST expression, it could be useful to specify what the result should be if the CAST would otherwise raise an exception. For example:

```
CAST ('abc' AS INT ON EXCEPTION NULL)
CAST ('abc' AS INT ON EXCEPTION 0)
CAST (a.prop1 AS INT ON EXCEPTION a.prop2)
```

An ON EXCEPTION capability would be particularly useful when loading large volumes of data.

Solution:

None provided with comment.

GQL-225 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 19.22, "<cast specification>".

Note At: None.

Source: W13-012

Language Opportunity:

<cast specification> does not currently support converting byte strings to character strings or character strings to byte strings (using some well-known encoding). Such a capability might be useful.

Solution:

None provided with comment.

GQL-226 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 19.22, "<cast specification>".

Note At: None.

Source: Editors.

Language Opportunity:

The ability to extract individual time scale components from temporal instances and duration would be useful and might be used to simplify the specification of <cast specification>. SQL has an EXTRACT function which satisfies a similar need and openCypher in its CIP2015-08-06 Date and Time specification has defined the ability to extract all individual time scale components.

Solution:

None provided with comment.

GQL-227 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 13.3, "<drop schema statement>".

Note At: None.

Source: W13-024.

Language Opportunity:

<drop schema statement> only allows dropping a schema if it does not contain any catalog object. More user convenience could potentially be added to the <drop schema statement> by including a new option for cascading drop statements or by developing a general invalidation model for catalog objects.

Solution:

None provided with comment.

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

GQL-229 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: None.

Source: W15-018.

Language Opportunity:

Pattern Macros could provide powerful capabilities for GQL Users by providing a multi-use template.
A Pattern Macro could be created in several places:

- As a preamble to a single query that lasts for the life of the query.
- As a catalog object that can be created, used in multiple queries, and lasts until explicitly dropped.

This capability should be considered for a future GQL version.

Solution:

None provided with comment.

GQL-230 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 16.18, “<order by clause>”.

Note At: None.

Source: W15-020.

Language Opportunity:

WITH ORDINALITY, WITH INDEX, WITH GROUP ORDINALITY and WITH GROUP INDEX to be added.

Solution:

None provided with comment.

GQL-231 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 18.9, “<source/destination predicate>”.

Note At: No specific location.

Source: WG3:W15-022.

Language Opportunity:

The <source/destination predicate> is expressed as a predicate about the node, while it is most often used to pose questions about the edge. If an alternative form of these predicates was available that put the edge as the first operand, this predicate would be more useful in combination with <simple case>.

Linked to Language Opportunity [PGQ-071](#).

Solution:

None provided with comment.

GQL-233 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 21.5, "Applying bindings to evaluate an expression".

Note At: Editor's Note number 270.

Source: MMX-035r2, WG3:W13-018.

Language Opportunity:

The bindings of a group reference flatten nested lists. This may be acceptable for SQL aggregates, which have no support for nested groupings, but may be inadequate to fully capture the semantics of a group reference in a graph pattern. MMX-035r2 section 4.1 "Desynchronized lists" pointed out a problem with reducing group variables to lists: two lists may be interleaved, but the reduction to separate lists can lose this information. The example given is

```
( (A:Person) -[:SPOUSE]-> ()  
| (B:Person) -[:FRIEND]-> () ){3}
```

A solution may find matches to A and B in any order. With separate lists of matches of A and B, it will not be easy to reconstruct the precise sequence of interleaved matches to A and B.

A similar problem can arise with nested quantifiers. MMX-035r2 section 4.2 "Nested quantifiers" gives this example:

```
( (C1:CORP) (-[ :TRANSFERS ]->(B:BANK) )*  
-[ :TRANSFERS ]-> (C2:CORP) )*
```

With this pattern, there can be 0 or more bindings to B between any two consecutive bindings to C1 and C2. With just independent lists of matches to C1, B and C2, it will not be easy to determine which bindings to B lie between which bindings to C1 and C2.

Linked to Language Opportunity **PGQ-069**.

Solution:

None provided with comment.

GQL-234 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 15.2, "<composite query expression>".

Note At: None.

Source: W15-017.

Language Opportunity:

Subclause 15.2, "<composite query expression>", General Rules are written to compose the binding tables generated by pairs of queries. There might be a benefit in expanding the GRs to also support operations directly on graphs such that GQL gains the capability to do, e.g., union of graphs.

Solution:

None provided with comment.

GQL-235 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

Note At: None.

Source: W16-038.

Language Opportunity:

The ability to create, reference, and drop constants as catalog objects could be very useful when constructing GQL queries. This capability should be considered for a future GQL version.

Solution:

None provided with comment.

GQL-237 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: None.

Source: W16-041.

Language Opportunity:

The ability to query a catalog to identify graphs with particular characteristics would be useful.

Solution:

None provided with comment.

GQL-241 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 13.12, “<property type set definition>”.

Note At: None.

Source: W19-019R1.

Language Opportunity:

Several future extensions of the GQL language (See [Language Opportunity GQL-241](#), [Language Opportunity GQL-242](#), [Language Opportunity GQL-243](#)) depend on properties being directly referenceable as graph elements in their own rights. Currently properties only exist (from a schema point of view) as members in property type sets attached to either nodes or edges. A definition point of a property type as such is needed. Such a graph element will be necessary for future concept relationships between properties and, e.g., other properties, other edges and other nodes than the ones that the current property set “belongs to”.

Solution:

None provided with comment.

GQL-242 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: [Language Opportunity GQL-241](#).

Note At: None.

Source: W19-019R1.

Language Opportunity:

Functional dependencies in a more specific manner (minimizing the need for inferencing in query planning) require explicit concept relationships between properties (e.g., keys and non-keys) for dealing with uniqueness and key formation.

Solution:

None provided with comment.

GQL-243 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Language Opportunity [GQL-241](#).

Note At: None.

Source: W19-019R1.

Language Opportunity:

Versioning (of both data and metadata) and timeline handling require explicit concept relationships between properties (e.g., keys and non-keys) for dealing with uniqueness and key formation.

Solution:

None provided with comment.

GQL-244 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Language Opportunity [GQL-241](#).

Note At: None.

Source: W19-019R1.

Language Opportunity:

Coexistence between LPG(s) of different breeds and also RDF require meta transformations between concept systems describing: a source graph type, a “universal” abstract graph type system and a target graph type. This requires GQL properties to be directly referenceable as such in combinations with other graph elements, including other properties, edges and relationships.

Solution:

None provided with comment.

GQL-245 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: None.

Source: W19-017.

Language Opportunity:

Allowing the bundling of multiple requests into one is useful for reducing network round trips. This should include the capability for returning multiple results to the client in one round trip.

Solution:

One possibility is to introduce a new top-level Subclause <GQL-multi-request> next to <GQL-request>.

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

GQL-246 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: None.

Source: WG3:W20-012.

Language Opportunity:

The ability to create, call, and drop named, stored procedures is a powerful capability when building applications in a GQL database. This capability should be considered for a future GQL version and should include both mandatory and optional parameters.

The last Informal Working Draft that contained a sketch of the functionality (deleted by W20-012) was 39075_1IWD21-GQL_2022-02 which is available here: https://sd.iso.org/documents/-ui/#!/browse/iso/iso-iec-jtc-1/iso-iec-jtc-1-sc-32/iso-iec-jtc-1-sc-32-wg-3/library/6/16656391_LL/16656048_LL/39075_1IWD21-GQL_2022-02.pdf

Solution:

None provided with comment.

GQL-247 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: None.

Source: WG3:W20-012.

Language Opportunity:

The ability to create, invoke, and drop named, stored queries is a powerful capability when building applications in a GQL database. This capability should be considered for a future GQL version.

The last Informal Working Draft that contained a sketch of the functionality (deleted by W20-012) was 39075_1IWD21-GQL_2022-02 which is available here: https://sd.iso.org/documents/-ui/#!/browse/iso/iso-iec-jtc-1/iso-iec-jtc-1-sc-32/iso-iec-jtc-1-sc-32-wg-3/library/6/16656391_LL/16656048_LL/39075_1IWD21-GQL_2022-02.pdf

Solution:

None provided with comment.

GQL-248 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: None.

Source: WG3:W20-013.

Language Opportunity:

The ability to create, call, and drop named, stored functions is a powerful capability when building applications in a GQL database. This capability should be considered for a future GQL version and should include both mandatory and optional parameters. As part of the consideration, there should

be a discussion of whether GQL should support the definition and/or calling of pure functions, impure functions, or both.

The last Informal Working Draft that contained a sketch of the functionality (deleted by W20-013) was 39075_1IWD21-GQL_2022-02 which is available here: https://sd.iso.org/documents/-ui/#!/browse/iso/iso-iec-jtc-1/iso-iec-jtc-1-sc-32/iso-iec-jtc-1-sc-32-wg-3/library/6/16656391_LL/16656048_LL/39075_1IWD21-GQL_2022-02.pdf

Solution:

None provided with comment.

GQL-250 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: Subclause 4.3.4, "Graphs".

Note At: None.

Source: WG3:RKE-040.

Language Opportunity:

GQL should be extended to support subgraphs for the most needed use cases.

Paper WG3:RKE-046 Subgraphs of property graphs provides some commentary on this topic.

Solution:

None provided with comment.

GQL-279 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference:

« WG3:BER-031 »

Subclause 4.11.5, "References to graph pattern variables".

Note At: Editor's Note number 25.

Source: WG3:BER-031.

Language Opportunity:

It is a Language Opportunity to support references to subpath variables, for example, in <graphical path length function>, or a TOTAL_COST function once CHEAPEST is defined.

Linked to Language Opportunity **PGQ-085**.

Solution:

None provided with comment.

GQL-281 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference:

« WG3:BER-026 »

Subclause 4.16.3.4.1, "Introduction to numbers".

Note At: None.

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

Source: WG3:BER-026.

Language Opportunity:

There is the opportunity to support the IEEE rounding modes.

Solution:

None provided with comment.

GQL-282 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: None.

Source: WG3:BER-026.

Language Opportunity:

SQL/PGQ has an IS BOUND predicate to test whether an element variable is bound (See WG3:W18-028). This predicate should be considered for GQL as an advanced conformance feature.

Solution:

None provided with comment.

GQL-283 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference:

[« WG3:BER-026 »](#)

No specific location.

Note At: None.

Source: WG3:BER-038R1.

Language Opportunity:

An ordered set collection type that provides an ordered collection of distinguishable elements might be useful for GQL users.

Solution:

None provided with comment.

GQL-285 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference:

[« WG3:BER-040R3 »](#)

No specific location.

Note At: None.

Source: WG3:BER-040R3.

Language Opportunity:

Support for additional syntactic forms currently ruled out for <edge reference value type> should be considered.

Solution:

None provided with comment.

GQL-286 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference:

« WG3:BER-049 »

No specific location.

Note At: None.

Source: WG3:BER-049.

Language Opportunity:

The ability to construct a conditional data modification statement with multiple when-then branches might be useful to GQL users.

Solution:

None provided with comment.

GQL-287 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference: No specific location.

Note At: None.

Source: WG3:BER-049.

Language Opportunity:

The ability to construct a conditional query with multiple when-then branches might be useful to GQL users.

Solution:

None provided with comment.

GQL-288 The following Language Opportunity has been noted:

Severity: Language Opportunity

Reference:

« WG3:BER-050 »

No specific location.

Note At: None.

Source: WG3:BER-050.

Language Opportunity:

The ability to MERGE a <simple graph pattern> into an existing graph would be useful to GQL users.

See BER-089, "Thoughts about MERGE", for a discussion of this topic.

Editor's Notes for IWD 39075:202y(E)
Language Opportunities

Solution:

None provided with comment.