INTERNATIONAL STANDARD

ISO/IEC 9075-13

Fifth edition
2023-06

Information technology — Database languages SQL —

Part 13:
SQL Routines and types using the Java TM programming language (SQL/JRT)

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

Partie 13: Routines et types de SQL utilisant le langage de programmation Java TM (SQL/JRT)

Reference number
ISO/IEC 9075-13:2023(E)

Contents

Page

Foreword . . . vii

Introduction . . . ix

1 Scope . . . 1

2 Normative references . . . 2

3 Terms and definitions . . . 3

4 Concepts . . . 6

4.1 Notations and conventions . . . 6

4.1.1 Notations . . . 6

4.1.2 Specification of built-in procedures . . . 6

4.1.3 Specification of deployment descriptor files . . . 6

4.2 Java programming language . . . 7

4.3 SQL-invoked routines . . . 8

4.3.1 Characteristics of SQL-invoked routines . . . 8

4.4 Java class name resolution . . . 9

4.5 SQL result sets . . . 10

4.6 Parameter mapping . . . 10

4.7 Unhandled Java exceptions . . . 11

4.8 Data types . . . 11

4.8.1 Host language data types . . . 11

4.9 User-defined types . . . 12

4.9.1 Introduction to user-defined types . . . 12

4.9.2 User-defined type comparison and assignment . . . 13

4.9.3 User-defined type descriptor . . . 13

4.9.4 Accessing static fields . . . 14

4.9.5 Converting objects between SQL and Java . . . 15

4.9.5.1 Introduction to conversion of objects between SQL and Java . . . 15

4.9.5.2 SERIALIZABLE . . . 15

4.9.5.3 SQLDATA . . . 15

4.9.5.4 Developing for portability . . . 16

4.10 Built-in procedures . . . 16

4.11 Basic security model . . . 16

4.11.1 Privileges . . . 16

4.12 JARs . . . 17

4.12.1 Introduction to JARs . . . 17

4.12.2 Deployment descriptor files . . . 17

5 Lexical elements . . . 19

5.1 <token> and <separator> . . . 19

5.2 Names and identifiers . . . 20

6 Scalar expressions 22
6.1 <method invocation> 22
6.2 <new specification> 23
7 Predicates 24
7.1 <comparison predicate> 24
8 Additional common rules 26
8.1 Invoking an SQL-invoked routine 26
8.2 Execution of array-returning functions 35
9 Additional common elements 42
9.1 <language clause> 42
9.2 <Java parameter declaration list> 43
9.3 <SQL Java path> 44
9.4 <routine invocation> 46
9.5 Java routine signature determination 47
10 Schema definition and manipulation 55
10.1 <drop schema statement> 55
10.2 <table definition> 56
10.3 <view definition> 57
10.4 <user-defined type definition> 58
10.5 <attribute definition> 62
10.6 <alter type statement> 65
10.7 <drop data type statement> 66
10.8 <SQL-invoked routine> 67
10.9 <alter routine statement> 70
10.10 <drop routine statement> 71
10.11 <user-defined ordering definition> 72
10.12 <drop user-defined ordering statement> 73
11 Access control 74
11.1 <grant privilege statement> 74
11.2 <privileges> 75
11.3 <revoke statement> 76
12 Built-in procedures 77
12.1 SQLJ.INSTALL_JAR procedure 77
12.2 SQLJ.REPLACE_JAR procedure 79
12.3 SQLJ.REMOVE_JAR procedure 81
12.4 SQLJ.ALTER_JAVA_PATH procedure 83
13 Java topics 85
13.1 Java facilities supported by this document 85
13.1.1 Package java.sql 85
13.1.2 System properties 85
13.2 Deployment descriptor files 86
14 Information Schema 89
14.1 Information Schema digital artifact 89
14.2 JAR_JAR_USAGE view 89
14.3 JARS view 90

14.4 METHOD_SPECIFICATIONS view. 91
14.5 ROUTINE_JAR_USAGE view. 92
14.6 TYPE_JAR_USAGE view. 93
14.7 USER_DEFINED_TYPES view. 94
14.8 Short name views. 95

15 Definition Schema. 96
15.1 Definition Schema digital artifact. 96
15.2 JAR_JAR_USAGE base table. 96
15.3 JARS base table. 98
15.4 METHOD_SPECIFICATIONS base table. 99
15.5 ROUTINE_JAR_USAGE base table. 101
15.6 ROUTINES base table. 102
15.7 TYPE_JAR_USAGE base table. 103
15.8 USAGE_PRIVILEGES base table. 104
15.9 USER_DEFINED_TYPES base table. 105

16 Status codes. 107
16.1 SQLSTATE. 107

17 Conformance. 108
17.1 Claims of conformance to SQL/JRT. 108
17.2 Additional conformance requirements for SQL/JRT. 108
17.3 Implied feature relationships of SQL/JRT. 108

Annex A (informative) **SQL conformance summary. 109**

Annex B (informative) **Implementation-defined elements. 115**

Annex C (informative) **Implementation-dependent elements. 121**

Annex D (informative) **SQL optional feature taxonomy. 123**

Annex E (informative) **Deprecated features. 125**

Annex F (informative) **Incompatibilities with ISO/IEC 9075:2016. . . . . 126**

Annex G (informative) **Defect Reports not addressed in this edition of this document. 127**

Bibliography. 128

Index. 129

Tables

Table **Page**

1 Standard programming languages 42
2 System properties 85
3 SQLSTATE class and subclass codes 107
4 Implied feature relationships of SQL/JRT 108
A.1 Feature definitions outside of Conformance Rules 109
D.1 Feature taxonomy for optional features 123

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).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC have not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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 fifth edition cancels and replaces the fourth edition (ISO/IEC 9075-13:2016), which has been technically revised. It also incorporates the Technical Corrigenda ISO/IEC 9075-13:2016/Cor.1:2019 and ISO/IEC 9075-13:2016/Cor.2:2022.

The main changes are as follows:

— improve the presentation and accuracy of the summaries of implementation-defined and implementation-dependent aspects of this document;

— introduction of several digital artifacts;

— alignment with updated ISO house style and other guidelines for creating standards.

This fifth edition of ISO/IEC 9075-13 is designed to be used in conjunction with the following editions of other parts of the ISO/IEC 9075 series, all published in 2023:

— ISO/IEC 9075-1, sixth edition;

— ISO/IEC 9075-2, sixth edition;

— ISO/IEC 9075-3, sixth edition;

— ISO/IEC 9075-4, seventh edition;

— ISO/IEC 9075-9, fifth edition;

— ISO/IEC 9075-10, fifth edition;

— ISO/IEC 9075-11, fifth edition;

— ISO/IEC 9075-14, sixth edition;

— ISO/IEC 9075-15, second edition;

— ISO/IEC 9075-16, first edition.

A list of all parts in the ISO/IEC 9075 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

The organization of this document is as follows:

1) Clause 1, “Scope”, specifies the scope of this document.

2) Clause 2, “Normative references”, identifies additional standards that, through reference in this document, constitute provisions of this document.

3) Clause 3, “Terms and definitions”, defines the terms and definitions used in this document.

4) Clause 4, “Concepts”, presents concepts used in the definition of Java routines and types.

5) Clause 5, “Lexical elements”, defines a number of lexical elements used in the definition of Java routines and types.

6) Clause 6, “Scalar expressions”, defines the elements of the language that produce scalar values.

7) Clause 7, “Predicates”, defines the predicates of the language.

8) Clause 8, “Additional common rules”, specifies the rules for assignments that retrieve data from or store data into SQL-data, and formation rules for set operations.

9) Clause 9, “Additional common elements”, defines additional language elements that are used in various parts of the language.

10) Clause 10, “Schema definition and manipulation”, defines the schema definition and manipulation statements associated with the definition of Java routines and types.

11) Clause 11, “Access control”, defines facilities for controlling access to SQL-data.

12) Clause 12, “Built-in procedures”, defines new built-in procedures used in the definition of Java routines and types.

13) Clause 13, “Java topics”, defines the facilities supported by implementations of this document and the conventions used in deployment descriptor files.

14) Clause 14, “Information Schema”, defines viewed tables that contain schema information.

15) Clause 15, “Definition Schema”, defines base tables on which the viewed tables containing schema information depend.

16) Clause 16, “Status codes”, defines SQLSTATE values related to Java routines and types.

17) Clause 17, “Conformance”, defines the criteria for conformance to this document.

18) Annex A, “SQL conformance summary”, is an informative Annex. It summarizes the conformance requirements of the SQL language.

19) Annex B, “Implementation-defined elements”, is an informative Annex. It lists those features for which the body of this document states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or other aspect is partly or wholly implementation-defined.

20) Annex C, “Implementation-dependent elements”, is an informative Annex. It lists those features for which the body of this document states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or other aspect is partly or wholly implementation-dependent.

21) Annex D, “SQL optional feature taxonomy”, is an informative Annex. It identifies the optional features of the SQL language specified in this document by an identifier and a short descriptive name. This taxonomy is used to specify conformance.

22) Annex E, “Deprecated features”, is an informative Annex. It lists features that the responsible Technical Committee intends not to include in a future edition of this document.

23) Annex F, “Incompatibilities with ISO/IEC 9075:2016”, is an informative Annex. It lists incompatibilities with the previous edition of this document.

24) Annex G, “Defect Reports not addressed in this edition of this document”, is an informative Annex. It describes the Defect Reports that were known at the time of publication of this document. Each of these problems is a problem carried forward from the previous edition of the ISO/IEC 9075 series. No new problems have been created in the drafting of this edition of this document.

In the text of this document, Clauses begin a new odd-numbered page, and in Clause 5, “Lexical elements”, through Clause 17, “Conformance”, Subclauses begin a new page. Any resulting blank space is not significant.

Information technology — Database language SQL —

Part 13:
SQL Routines and Types Using the Java™ Programming Language (SQL/JRT)

1 Scope

This document specifies the ability to invoke static methods written in the Java[1] programming language as SQL-invoked routines and to use classes defined in the Java programming language as SQL structured user-defined types.

1 Java™ is a registered trademark of Oracle Corporation and/or its affiliates.

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/IEC 9075-1, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*

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

ISO/IEC 9075-10, *Information technology — Database languages — SQL — Part 10: Object Language Bindings (SQL/OLB)*

ISO/IEC 9075-11, *Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata)*

Java Community Process. *The Java™ Language Specification* [online]. Java SE 13 Edition. Redwood Shores, California, USA: Oracle, Available at https://docs.oracle.com/javase/specs/jls/se13/jls13.pdf

Java Community Process. *The Java™ Virtual Machine Specification* [online]. Java SE 13 Edition. Redwood Shores, California, USA: Oracle, Available at https://docs.oracle.com/javase/specs/jvms/se13/jvms13.pdf

Java Community Process. *Java™ Object Serialization Specification* [online]. Redwood Shores, California, USA: Oracle, Available at https://docs.oracle.com/en/java/javase/13/docs/specs/serialization/serial-arch.html

Java Community Process. *JDBC™ 4.3 Specification* [online]. Edition 4.3. Redwood Shores, California, USA: Oracle, Available at https://download.oracle.com/otn-pub/jcp/jdbc-4_3-mrel3-eval-spec/jdbc4.3-fr-spec.pdf

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 9075-1, ISO/IEC 9075-2, Java, JVMS, and the following apply.

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

— ISO Online browsing platform: available at `https://www.iso.org/obp`

— IEC Electropedia: available at `https://www.electropedia.org/`

3.1
block
related statements within braces, which would be treated as a single statement

3.2
class declaration
specifies a possibly empty set consisting of zero or more *fields* (3.6), zero or more methods, zero or more *nested classes* (3.11), zero or more *interfaces* (3.9), zero or more *instance initializers* (3.7), zero or more *static initializers* (3.13), and zero or more constructors

3.3
class file
file containing Java byte code

3.4
class instance
consists of an instance of each *instance variable* (3.8) declared in the class and each *instance variable* (3.8) declared in the superclasses of the class

3.5
class variable
static variable declared in a class

3.6
field
class variables (3.5) and *instance variables* (3.8) of a class

3.7
instance initializer
block that is used to initialize the instance data member, that runs each time when object of the class is created

3.8
instance variable
variable that is declared inside a class but outside the scope of any method

3.9
interface
Java construct consisting of a set of method signatures

3.10
local variable
variables declared inside the body of a method

3.11
nested class
class within another class

3.12
package
zero or more classes, zero or more *interfaces* (3.9), and zero or more *subpackages* (3.14)

3.13
static initializer
executes code before the object initialization

3.14
subpackage
package (3.12) within a *package* (3.12)

3.15
JVM
Java Virtual Machine
virtual machine that enables a computer to run Java programs

3.16
default connection
JDBC connection to the current SQL-implementation, SQL-session, and SQL-transaction established with the data source URL `'jdbc:default:connection'`

Note 1 to entry: See RFC 2368 and RFC 3986 for more details about URLs.

3.17
deployment descriptor
one or more SQL-statements that specify <install actions> and <remove actions> to be taken, respectively, by the `SQLJ.INSTALL_JAR` and `SQLJ.REMOVE_JAR` procedures and that are contained in a *deployment descriptor file* (3.18)

3.18
deployment descriptor file
text file containing *deployment descriptors* (3.17) that is contained in a *JAR* (3.22), for which the JAR's manifest entry, as described by the `java.util.jar` section of JVMS, specifies `SQLJDeploymentDescriptor: TRUE`

3.19
external Java data type
SQL user-defined type defined with a <user-defined type definition> that specifies an <external Java type clause>

3.20
external Java routine
external routine defined with an <SQL-invoked routine> that specifies LANGUAGE JAVA and either PROCEDURE or FUNCTION, or defined with a <user-defined type definition> that specifies an <external Java type clause>

3.21
installed JAR
JAR (3.22) whose existence has been registered with the SQL-environment and whose contents have been copied into that SQL-environment due to execution of one of the procedures `SQLJ.INSTALL_JAR` and `SQLJ.REPLACE_JAR`

3.22
JAR
Java Archive
zip-formatted file, as described by the `java.util.zip` section of JVMS, containing zero or more Java `class` and `ser` files, and zero or more deployment descriptor files

Note 1 to entry: JARs are a normal vehicle for distributing Java programs and the mechanism specified by this document to provide the implementation of external Java routines and external Java data types to an SQL-environment.

3.23
ser file
file containing representations of Java objects in the form defined in Java Serialization

3.24
subject Java class
Java class uniquely identified by the combination of the class's *subject Java class name* (3.25) and its containing *JAR* (3.22)

3.25
subject Java class name
fully-qualified package name and class name of a Java class

3.26
system class
Java class provided by a conforming implementation of this document that can be referenced by an external Java routine or an external Java data type without that class having been included in an installed *JAR* (3.22)

4 Concepts

This Clause modifies Clause 4, “Concepts”, in ISO/IEC 9075-2.

4.1 Notations and conventions

This Subclause modifies Subclause 4.1, “Notations and conventions”, in ISO/IEC 9075-2.

4.1.1 Notations

This Subclause modifies Subclause 4.1.1, “Notations”, in ISO/IEC 9075-2.

The notations used in this document are defined in ISO/IEC 9075-1.

The syntax defined in this document is available from the ISO website as a “digital artifact”. See `https://standards.iso.org/iso-iec/9075/-13/ed-5/en/` to download digital artifacts for this document. To download the syntax defined in a plain-text format, select the file named `ISO_IEC_9075-13(E)_JRT.bnf.txt`. To download the syntax defined in an XML format, select the file named `ISO_IEC_9075-13(E)_JRT.bnf.xml`.

4.1.2 Specification of built-in procedures

Built-in procedures are specified in terms of the following information.

— **Function**: A short statement of the purpose of the procedure.

— **Signature**: A specification, in SQL, of the signature of the procedure. The only purpose of the signature is to specify the procedure name, parameter names, and parameter types. The manner in which these built-in procedures are defined is implementation-dependent (UW004).

— **Access Rules**: A specification in English of rules governing the accessibility of schema objects that shall hold before the General Rules may be successfully applied.

— **General Rules**: A specification in English of the run-time effect of invocation of the procedure. 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 SQL.

The scope of notational symbols is the Subclause in which those symbols are defined. Within a Subclause, the symbols defined in the Signature, Access Rules, or General Rules can be referenced in other rules provided that they are defined before being referenced.

4.1.3 Specification of deployment descriptor files

Deployment descriptor files are specified in terms of the following.

— **Function**: A short statement of the purpose of the deployment descriptor file.

— **Model**: A brief description of the manner in which a deployment descriptor file is identified within its containing JAR.

— **Properties**: A BNF specification of the syntax of the contents of a deployment descriptor file.

— **Description**: A specification of the requirements and restrictions on the contents of a deployment descriptor file.

— **Conformance Rules**: A specification of how the element shall be supported for conformance to SQL.

4.2 Java programming language

The Java programming language is a class-based, object-oriented language. This document uses the following Java concepts and terminology.

A *class* is the basic construct of Java programs, in that all executable Java code is contained in a Java class definition. A class is declared by a *class declaration* that specifies a possibly empty set consisting of zero or more fields, zero or more methods, zero or more nested classes, zero or more interfaces, zero or more instance initializers, zero or more static initializers, and zero or more constructors.

The scope of a variable is a class, an instance of the class, or a method of the class. The scope of a variable that is declared *static* is the class, and the variable is called a *class variable*. The scope of each other variable declared in the class is instances of the class, and such a variable is called an *instance variable*. Class variables and instance variables of a class are called *fields* of that class. The scope of a variable declared in a method is the *block* or Java `for` statement in which it is declared in that method, and the variable is called a *local variable*.

A *class instance* consists of an instance of each instance variable declared in the class and each instance variable declared in the superclasses of the class. Class instances are strongly typed by the class name. The operations available on a class instance are those defined for its class.

With the exception of the class `java.lang.Object`, each class is declared to *extend* (at most) one other class; a class not explicitly declared to extend another class implicitly extends `java.lang.Object`. The declared class is a *direct subclass* of the class that it extends; the class that it extends is the *direct superclass* of the declared class.

Class *B* is a *subclass* of class *A* if *B* is a direct subclass of *A*, or if there exists some class *C* such that *B* is a direct subclass of *C* and *C* is a subclass of *A*. Likewise, class *B* is a *superclass* of class *A* if *B* is a direct superclass of *A*, or if there exists some class *C* such that *B* is a direct superclass of *C* and *C* is a superclass of *A*. A subclass has all of the fields and methods of its superclasses and an instance of class *B* may be used wherever an instance of a superclass of *B* is permitted.

A *method* is an executable routine. A method can be declared *static*, in which case it is called a *class method*; otherwise, it is called an *instance method*. A class method can be referenced by qualifying the method name with either the class name or the name of an instance of the class. An instance method is referenced by qualifying the method name with a Java expression that results in an instance of the class or, in the case of a constructor, with "`new`". The method body of an instance method can reference its class variables, instance variables, and local variables.

The *Java method signature* of a method consists of the method name and the number of parameters and their data types.

A *package* consists of zero or more classes, zero or more interfaces, and zero or more *subpackages* (a subpackage is a package within a package); each package provides its own namespace and classes within a package are able to refer to other classes in the same package, including classes not referenceable from outside the package. Every class belongs to exactly one package, either an explicitly specified named package or the anonymous default package. A class can specify Java `import` statements to refer to other named packages whose classes can then be referenced within the class without package qualification.

A class, field, or methods can be declared as *public*, *private*, or *protected*. A public variable or method can be accessed by any method. A private variable or method can only be referenced by methods in the same class. A protected variable or method can only be referenced by methods of the same class or subclasses thereof. A method that is not declared as public, private, or protected can only be called by methods declared by classes in the same package.

An *interface* is a Java construct consisting of a set of method signatures. An interface can be implemented by zero or more classes, a class can be declared to implement zero or more interfaces, and a class is required to have methods with the signatures specified by all of its declared interfaces.

The Java *Serializable* interface, `java.io.Serializable`, as described in JVMS, defines a transformation between a Java instance and a `java.io.OutputStream` or `java.io.InputStream`, as defined by the `java.io.OutputStream` and `java.io.InputStream` sections of JVMS respectively, writing a persistent representation of an instance of a Java object and reading a persistent representation of an instance of a Java object. This transformation retains sufficient information to identify the most specific class of the instance and to reconstruct the instance.

The Java *SQLData* interface, `java.sql.SQLData`, as described in JDBC and JVMS, defines a transformation between a Java instance and an SQL user-defined data type.

The source for a Java class is normally stored in a *Java file* with the file-type "java", e.g., `myclass.java`. Java is normally compiled to a byte coded instruction set that is portable to any platform supporting Java. A file containing such byte code is normally stored in a *class file* with the file-type "class", e.g., `myc-lass.class`.

A set of class files can be assembled into a *Java archive* file, or *JAR* (usually with a file extension of ".jar". A JAR is a zip formatted file containing a set of Java class files. JARs are the normal vehicle for distributing Java programs.

4.3 SQL-invoked routines

This Subclause modifies Subclause 4.35, "SQL-invoked routines", in ISO/IEC 9075-2.

4.3.1 Characteristics of SQL-invoked routines

This Subclause modifies Subclause 4.35.2, "Characteristics of SQL-invoked routines", in ISO/IEC 9075-2.

Insert after the 2nd paragraph: External routines appear in two seemingly similar, but fundamentally differing, forms, where the key differentiator is whether or not the external routine's routine descriptor specifies that the body of the SQL-invoked routine is written in Java. When the body of the SQL-invoked routine is written in Java, the external routine is an *external Java routine*; some differences from other external routines include the following items.

— For any other external routine, the *executable form* (such as a dynamic link library or some run-time interpreted form) of that routine exists externally to the SQL-environment's catalogs; for an external Java routine, the executable form is provided by a specified subject Java routine that exists in the SQL-environment's catalogs in an installed JAR.

— Because an installed JAR is not required to be completely self-contained (i.e., to have no references to Java classes outside of itself), a mechanism is provided to allow a subject Java class to reference classes defined by class files contained in its installed JAR or in other installed JARs. See Subclause 9.3, "<SQL Java path>".

NOTE 1 — Once an external Java routine has been created, its use in SQL statements executed by the containing SQL-environment is similar to that of other external routines.

Insert into the 8th paragraph, after the last list item:

— If the instance SQL-invoked method is an external Java routine, the term "set of overriding methods" is not applicable; for such methods, the capabilities provided by overriding methods duplicate Java's own mechanisms and the subject routine executed is the one that would be invoked when no overriding methods are specified.

Augment the 15th paragraph by adding “When the SQL-invoked routine is not an external Java routine,” as a restriction to the beginning of the Rule.

Insert after the 15th paragraph: When the SQL-invoked routine is an external Java routine, values in the effective SQL parameter list are passed to the program identified by <routine body> according to the rules of Subclause 4.6, “Parameter mapping”.

4.4 Java class name resolution

Typical JVMs provide a *class name resolution*, or search path, mechanism based on an environmental variable called CLASSPATH. When a JVM encounters a previously unseen reference to a class, the members of the list of directories and JARs appearing in the classpath are examined in order until either the class is found or the end of the list is reached. Failure to locate a referenced class is a runtime error that will often cause the application that experiences it to terminate.

When JARs appear in the CLASSPATH, an ability exists for further effective extension of that CLASSPATH. Additional JARs will be included in the class resolution process when a JAR in the CLASSPATH has a manifest specifying one or more `Class-Path` attributes. A `Class-Path` attribute provides *relative URLs* of additional JARs. These `Class-Path` attribute URLs are relative to the source, for example, the directory, containing the JAR whose manifest is then being processed. A full URL, for example a `file:/` or `http://` format URL, is not allowed in a `Class-Path` attribute. The JARs enumerated by `Class-Path` attributes extend the CLASSPATH.

When a JVM is transitioned to being effectively within an SQL-environment, the problem of managing the JVM’s class name resolution continues to exist, but with a change in emphasis. One important change is that an installed JAR manifest’s `Class-Path` attributes cannot be honored. No relative URL has meaning when the source of the current JAR is given by a <catalog name>, <unqualified schema name>, and <jar id>. To allow the creators of Java applications a greater degree of control over class name resolution, and the added security associated with that control, a `Class-Path` attribute-like mechanism is defined to be a property of the JARs containing the Java applications, rather than as an environmental variable of the current session (such as, for example, CURRENT_PATH for dynamic statements). This mechanism, referred to as a JAR’s *SQL-Java path*, provides a means for owners of installed JARs to control the class resolution process that the CLASSPATH and `Class-Path` attributes give users and creators of JARs outside an SQL-environment. But, note that these two mechanisms are only similar, they are not identical. If, while an external Java routine is being executed, a previously unseen class reference is encountered, that class is searched for in the JAR containing the definition of the currently executing class, and, if it is not found, the class will be sought in the manner specified by the SQL-Java path associated with that JAR (if any).

An SQL-Java path specifies how a JVM resolves a class name when a class within a JAR references a class that is not a system class or not in the same JAR. `SQLJ.ALTER_JAVA_PATH` is used to associated an SQL-Java path with a JAR. An SQL-Java path is a list of (referenced item, referenced JAR) pairs. A referenced item can be either a class, a package, or '*' to specify the entire JAR. The SQL-Java path list is searched in the order the pairs are specified. For each (referenced item, referenced JAR) pair (*RI*, *RJ*), the following steps are taken.

— If *RI* is the class name, then the class shall be defined in *RJ*. If it is not, an exception condition is raised.

— If *RI* is the package of the class being resolved, then the class shall be defined in *RJ*. If it is not, an exception condition is raised.

— If *RI* is '*' and the class is defined in *RJ*, then that resolution is used; otherwise, subsequent pairs are tested.

4.5 SQL result sets

Cursors, or SQL result sets, appear to Java applications in two forms; the first, as an object of a class that implements the interface `java.sql.ResultSet` as defined in JDBC and JVMS, referred to as a *JDBC ResultSet*; the second, as an object of a class that implements the interface `sqlj.runtime.ResultSetIterator` as defined by ISO/IEC 9075-10, referred to as an *SQLJ Iterator*.

In ISO/IEC 9075-2, SQL-invoked procedures are declared to be able to return zero or more dynamic result sets, referred to as *result set cursors*. To be a returned result set cursor, a cursor's declaration shall specify WITH RETURN, and the cursor shall be open at the point that the SQL-invoked procedure exits. While external Java routines that are SQL-invoked procedures can likewise be declared to return zero or more dynamic result sets, in some other respects, this document's treatment of result set cursors differs from that of ISO/IEC 9075-2.

In a Java application, all JDBC ResultSets and SQLJ Iterators are implicitly result set cursors, that is, their underlying cursor declarations implicitly specify WITH RETURN. So, in this document, to actually be a returned result set cursor it is not sufficient that the corresponding JDBC ResultSet's or SQLJ Iterator's underlying cursor be open when the SQL-invoked procedure exits; the JDBC ResultSet or SQLJ Iterator shall also have been explicitly assigned to a parameter of the subject Java routine that represents an output parameter. As discussed in Subclause 4.6, "Parameter mapping", and Subclause 8.1, "Invoking an SQL-invoked routine", output parameters are represented to a subject Java routine as the first element of a one dimensional array of a Java data type that can be mapped to an SQL data type. For dynamic result sets, the array shall be of a class that implements the interface `java.sql.ResultSet` or the interface `sqlj.runtime.ResultSetIterator`, the JDBC ResultSet or SQLJ Iterator shall have been explicitly assigned to the first element of that array, and that JDBC ResultSet or SQLJ Iterator shall not have been closed.

It is important to note that this difference in how a result set cursor becomes a returned result set cursor is invisible to the calling application. As described in Subclause 8.1, "Invoking an SQL-invoked routine", the calling application will be returned zero or more dynamic result sets in the order in which the cursors were opened, just as in ISO/IEC 9075-2; the order of the parameters in the subject Java routine does not impact the order in which the calling application accesses the returned result sets.

4.6 Parameter mapping

Let *ST* be some SQL data type and let *JT* be some Java data type.

ST and *JT* are *simply mappable* if and only if *ST* and *JT* are specified respectively in the first and second columns of some row of the *Data type conversion tables*, Table B.1, entitled "JDBC Types mapped to Java Types", in JDBC. The Java data type *JT* is the *corresponding Java data type* of *ST*.

ST and *JT* are *object mappable* if and only if *ST* and *JT* are specified respectively in the first and second columns of some row of the *Data type conversion tables*, Table B.3, entitled "Mapping from JDBC Types to Java ObjectTypes", in JDBC, or if the descriptor of *ST* specifies that it is an external Java data type and the descriptor specifies *JT* as the <Java class name> in the <jar and class name>.

ST and *JT* are *output mappable* if and only if exactly one of the following is true.

— *JT* is a one dimensional array type with an element data type *JT2* (that is, *JT* is "*JT2*[]") and *ST* is either simply mappable to *JT2* or object mappable to *JT2*.

— *JT* is `java.lang.StringBuffer` and its corresponding parameter in the augmented SQL parameter declaration list is the save area data item.

An SQL array type with an element data type *ST* and *JT* are *array mappable* if and only if *JT* is a one dimensional array type with an element data type *JT2* and *ST* is either simply mappable to *JT2* or object mappable to *JT2*.

ST and *JT* are *mappable* if and only if *ST* and *JT* are simply mappable, object mappable, output mappable, or array mappable.

A Java data type is *mappable* if and only if it is mappable to some SQL data type.

A Java class is *result set oriented* if and only if it is either of the following.

— A class that implements the Java interface `java.sql.ResultSet`.

— A class that implements the Java interface `sqlj.runtime.ResultSetIterator`.

NOTE 2 — These classes are generated by iterator declarations (`#sql iterator`) as specified in ISO/IEC 9075-10.

A Java data type is *result set mappable* if and only if it is a one-dimensional array type with an element type that is a result set oriented class.

A Java method with *M* parameters is *mappable* (to SQL) if and only if, for some *N*, 0 (zero) ≤ *N* ≤ *M*, the data types of the first *N* parameters are mappable, the last *M*–*N* parameters are result set mappable, and the result type is either simply mappable, object mappable,or `void`.

A Java method is *visible* in SQL if and only if it is public and mappable. In addition, to be visible, a Java method shall be static if used as the external Java routine of an SQL-invoked procedure or an SQL-invoked regular function.

A Java class is *visible* in SQL if and only if it is public and mappable.

JDBC contains JDBC's SQL to Java data type mappings defined in the JDBC type mapping tables. If *ST* is an external Java data type that appears in the INFORMATION_SCHEMA.USER_DEFINED_TYPES view, then let *JT* be *ST*'s descriptor's <Java class name> in its <jar and class name>. JDBC's data type mapping tables are effectively extended. A row (*ST*, *JT*) is considered to be an additional row in Table B.3, *Mapping from JDBC Types to Java Object Types*, and a row (*JT*, *ST*) is considered to be an additional row in Table B.4, *Mapping from Java Object Types to JDBC Types* (both tables appear in JDBC).

4.7 Unhandled Java exceptions

Java exceptions that are thrown during execution of a Java method in SQL can be caught, or handled, within Java; if this is done, then those exceptions do not affect SQL processing. All Java exceptions that are uncaught when a Java method called from SQL completes appear in the SQL-environment as SQL exception conditions.

The message text may be specified in the Java exception specified in the Java `throw` statement. If the Java exception is an instance of `java.sql.SQLException`, or a subtype of that type, then it may also specify an SQLSTATE value. If the Java exception is not an instance of `java.sql.SQLException`, or if that exception does not specify an SQLSTATE value, then the default SQL exception condition for an uncaught Java exception is raised.

When a Java method executes an SQL statement, any exception condition raised in the SQL statement will be raised in the Java method as a Java exception that is specifically the `java.sql.SQLException` subclass of the Java class `java.lang.Exception`. For portability, a Java method called from SQL, that itself executes an SQL statement and that catches an SQLException from that inner SQL statement, should re-throw that SQLException.

4.8 Data types

This Subclause modifies Subclause 4.2, "Data types", in ISO/IEC 9075-2.

4.8.1 Host language data types

This Subclause modifies Subclause 4.2.3, "Host language data types", in ISO/IEC 9075-2.

Insert into the 1st paragraph, after the last list item:

— Subclause 11.1, "<embedded SQL host program>", in ISO/IEC 9075-10

4.9 User-defined types

This Subclause modifies Subclause 4.9, "User-defined types", in ISO/IEC 9075-2.

4.9.1 Introduction to user-defined types

This Subclause modifies Subclause 4.9.1, "Introduction to user-defined types", in ISO/IEC 9075-2.

Insert after the 1st paragraph: User-defined types appear in two seemingly similar, but fundamentally differing, forms in which the key differentiator is whether or not the create type statement for the user-defined type specifies an external language of "JAVA". When an external language of JAVA is specified, the user-defined type is an *external Java data type* and the create type statement defines a mapping of the user-defined type's attributes and methods directly to the public attributes and methods of a *subject Java class*. This is different from user-defined types that are not external Java data types. The differences include the following.

— For every other user-defined type, there is no requirement for an association with an underlying class; each method of a user-defined type that is not an external Java data type can be written in a different language (for example, one method could be written in SQL and another written in Fortran). Such user-defined types cannot have methods written in Java. By contrast, all methods of an external Java data type shall be written in Java, (implicitly) have a parameter style of JAVA, and be defined in the associated Java class or one of its superclasses.

— For every other user-defined type, there is no explicit association between a user-defined type's attributes and any external representation of their content. In addition, the mapping between a user-defined type's methods and external methods is made over time by subsequent CREATE METHOD statements. By contrast, for external Java data types, the association between the user-defined type's attributes and methods and the public attributes and methods of a subject Java class is specified by the create type statement.

— For external Java data types, the mechanism used to convert the SQL-environment's representation of an instance of a user-defined type into an instance of a Java class is specified in the <interface using clause>. Such conversions are performed, for example, when an external Java data type is specified as a (subject) parameter in a method or function invocation, or when a Java object returned from a method or function invocation is stored in a column declared to be an external Java data type. <interface specification> can be either SERIALIZABLE, specifying the Java-defined interface `java.io.Serializable` (not to be confused with the isolation level of SERIALIZABLE), or SQLDATA, specifying the JDBC-defined interface `java.sql.SQLData`. See Subclause 10.4, "<user-defined type definition>".

— For every other user-defined type, there is no explicit support of static attributes. For external Java data types, the <user-defined type definition> is allowed to include <static field method spec>s that define observer methods against specified static attributes of the subject Java class.

  The scope and persistence of any modifications to static attributes made during the execution of a Java method is implementation-dependent (UA003).

— For every other user-defined type, the implementation of every method that is not an SQL routine exists externally to the SQL-environment. For external Java data types, the implementation of the methods is provided by a specified subject Java class that exists within the SQL-environment in an *installed JAR*.

— External Java data types may only be structured types, not distinct types.

— Support for the specification of overriding methods is not provided for methods that are external Java routines.

NOTE 3 — Once an external Java data type has been created, its use in SQL statements executed by the containing SQL-implementation is similar to that of other user-defined types.

4.9.2 User-defined type comparison and assignment

This Subclause modifies Subclause 4.9.5, "User-defined type comparison and assignment", in ISO/IEC 9075-2.

Augment the 5th paragraph by adding "if any" at the end of the paragraph.

Insert into the 6th paragraph, after the last list item:

— If the comparison category is COMPARABLE, then no comparison functions shall be specified for *T1* and *T2*.

4.9.3 User-defined type descriptor

This Subclause modifies Subclause 4.9.7, "User-defined type descriptor", in ISO/IEC 9075-2.

Insert into the 1st paragraph, in the 4th list item, after the last list item:

— COMPARABLE

Insert into the 1st paragraph, after the 10th list item:

— An indication of whether the user-defined type is an external Java data type.

Insert after the 1st paragraph: If the user-defined type is an external Java data type, then the user-defined type descriptor also includes the following.

— The <jar and class name> of the user-defined type.

— The <interface specification> of SERIALIZABLE or SQLDATA.

— The attribute descriptor of every originally-defined attribute and every inherited attribute of the user-defined type.

— If <method specification list> is specified, then, for each <method specification> contained in <method specification list>, a *method spec descriptor* that includes the following.

- The <method name>.
- The <specific method name>.
- The <SQL parameter declaration list>.
- The <returns data type>, and indication of SELF AS RESULT.
- The <result cast from type>, if any.
- The package, class, and name of the Java routine corresponding to this method and, if specified, its signature.
- An indication of whether STATIC or CONSTRUCTOR is specified.
- If STATIC is specified, then an indication of whether this is a static field method.
- If this is a static field method, then the <Java field name> of the static field and the <Java class name> of the class that declares that static field.
- An indication of whether the method is deterministic.

- An indication of whether the method possibly writes SQL data, possibly reads SQL data, possibly contains SQL, or does not possibly contain SQL.
- An indication of whether the method is to be invoked if any argument is the null value, in which case the value of the method is the null value.

If the user-defined type is not an external Java data type, then the user-defined type descriptor also includes the following.

— An indication of whether the user-defined type is a structured type or a distinct type.

— If the representation is a predefined data type, then the descriptor of that type; otherwise, the attribute descriptor of every originally-defined attribute and every inherited attribute of the user-defined type.

— If the <method specification list> is specified, then, for each <method specification> contained in <method specification list>, a *method specification descriptor* that includes:

- The <method name>.
- The <specific method name>.
- The <SQL parameter declaration list> augmented to include the implicit first parameter with parameter name SELF.
- The <language name>.
- If the <language name> is not SQL, then the <parameter style>.
- The <returns data type>.
- The <result cast from type>, if any.
- An indication as to whether the <method specification> is an <original method specification> or an <overriding method specification>.
- If the <method specification> is an <original method specification>, then an indication of whether STATIC or CONSTRUCTOR is specified.
- An indication whether the method is deterministic.
- An indication whether the method possibly writes SQL data, possibly reads SQL data, possibly contains SQL, or does not possibly contain SQL.
- An indication whether the method is to be invoked if any argument is the null value, in which case the value of the method is the null value.

NOTE 4 — The characteristics of an <overriding method specification> other than the <method name>, <SQL parameter declaration list>, and <returns data type> are the same as the characteristics for the corresponding <original method specification>.

4.9.4 Accessing static fields

The fields of a Java class can be defined to be either *static* or *non-static*. Static fields of a Java class can additionally be specified to be *final*, which makes them read-only. In Java, non-final fields are allowed to be updated.

SQL's <user-defined type definition> does not include a facility for specifying attributes to be STATIC. This is, in part, because of the difficulty in specifying the scope, persistence, and transactional properties of static attributes in a database environment. An external Java data type's <user-defined type definition> does, however, provide a mechanism for read-only access to the values of Java static fields. The <static field method spec> clause defines a method name for a method with no parameters; its <external variable name clause> specifies the name of a static field of the subject Java class or a superclass of the subject

Java class. A static field method is invoked in the normal manner for STATIC methods and returns the value of the specified Java static field. Whether final or non-final, SQL provides no mechanism for updating the values of Java static fields.

4.9.5 Converting objects between SQL and Java

4.9.5.1 Introduction to conversion of objects between SQL and Java

While application programmers or end users manipulating Java objects in the database through SQL statements need not be aware of the specific mechanism used to achieve that conversion, the developer of the Java class itself needs to prepare for it in the form of implementing special Java interfaces (i.e., `java.io.Serializable` or `java.sql.SQLData`). <user-defined type definition> introduces a clause for specifying the interface for converting object state information between the SQL database and Java in the scope of SQL statements. As mentioned above, a conversion from SQL to Java can potentially take place when an object that has been persistently stored in the SQL database is accessed from inside an SQL statement to retrieve attribute (or field) values, or to invoke a method on the object, or when the object is used as an input argument in the invocation of a method. A conversion in the opposite direction, from Java to SQL, may be required when a newly created or modified object, or an object that is the return value of a method invocation, needs to be persistently stored in the database.

This document supports the following options to specify object state conversion in the <external Java type clause>.

— If the <user-defined type definition> specifies an <interface specification> of SERIALIZABLE, then the Java interface `java.io.Serializable` is used for object state conversion.

— If the <user-defined type definition> specifies an <interface specification> of SQLDATA, then the Java interface `java.sql.SQLData` defined in JDBC and JVMS is used for object state conversion.

— If the <user-defined type definition> does not specify an <interface specification>, then it is implementation-defined (IA132) whether the Java interface `java.io.Serializable` or the Java interface `java.sql.SQLData` will be used for object state conversion.

4.9.5.2 SERIALIZABLE

If the <interface specification> of a <user-defined type definition> specifies SERIALIZABLE, then object state communication is based on the Java interface `java.io.Serializable`. The Java class referenced in the <external Java class clause> of the <user-defined type definition> shall specify "`implements java.io.Serializable`" and shall provide a niladic constructor.

In this case, the SQL object state that is stored persistently and made available to methods of the SQL type is defined entirely by the Java serialized object state. The attributes defined for the SQL type shall correspond to public fields of the corresponding Java class, which shall be listed in the <external Java attribute clause> of each attribute. Consequently, the SQL attributes define access to those portions of the object state that are intended to become visible inside SQL statements, but might not comprise the complete state of the object (which may include additional fields in the Java class).

4.9.5.3 SQLDATA

If the <interface specification> of a <user-defined type definition> specifies SQLDATA,then object state communication is based on the Java interface `java.sql.SQLData` defined in JDBC and JVMS. The Java class referenced in the <external Java class clause> of the <user-defined type definition> shall specify "`implements java.sql.SQLData`" and shall provide a niladic constructor.

In this case, only the attributes defined in the statement comprise the complete state of the SQL object type. Additional public or private attributes defined in the Java class do not become part of the object state defined by this document. The Java object representation may be entirely different from the SQL object attributes, if desired. For example, an SQL Point type may define a geometric point in terms of

Cartesian coordinates, while the corresponding Java class defines it using polar coordinates. The only requirement to be met by the implementor of the Java class is that the implementations of the `java.sql.SQLData` methods `readSQL` and `writeSQL` read and write the attributes in the same order in which they are defined in the <user-defined type definition>.

To improve portability, it is possible to also specify <external Java attribute clause>s for SQL attributes, even if an <interface specification> of SQLDATA is specified. However, the <external Java attribute clause>s are ignored in this case, because they are not needed for implementing attribute access in SQL or for converting objects between SQL and Java.

4.9.5.4 Developing for portability

The following recommendations provide maximum portability of Java classes across different implementations of this document that may not support both the SERIALIZABLE and the SQLDATA options.

— The Java class used for implementing the SQL type should implement both `java.io.Serializable` and `java.sql.SQLData`.

— The Java class should define the complete object state that needs to become persistent or has to be preserved across invocations as public Java fields.

— The EXTERNAL NAMEs of the SQL attributes should be specified.

The <interface using clause> should be omitted in the <user-defined type definition>, so that an implementation that does not support both interfaces can default to the interface that it supports.

4.10 Built-in procedures

This document differs slightly from other parts of the ISO/IEC 9075 series in its treatment of the schema object introduced to install the external Java routines and external Java data types in an SQL-environment — that is, in its treatment of JARs. Rather than define new SQL-schema statements that (for example) add or drop JARs using optional clauses to cause execution of their contained deployment descriptors, this document introduces a set of four built-in procedures and a new schema in which those procedures are defined.

The new schema — named SQLJ — is, like the schema named INFORMATION_SCHEMA, defined to exist in all catalogs of an SQL system that implements this document, and to contain all of the built-in procedures defined in this document.

Built-in procedures defined in this document are the following.

— `SQLJ.INSTALL_JAR` — to load a set of Java classes in an SQL system.

— `SQLJ.REPLACE_JAR` — to supersede a set of Java classes in an SQL system.

— `SQLJ.REMOVE_JAR` — to delete a previously installed set of Java classes.

— `SQLJ.ALTER_JAVA_PATH` — to specify a path for name resolution within Java classes.

4.11 Basic security model

This Subclause modifies Subclause 4.42, "Basic security model", in ISO/IEC 9075-2.

4.11.1 Privileges

This Subclause modifies Subclause 4.42.2, "Privileges", in ISO/IEC 9075-2.

Insert into the 1st paragraph, after the last list item:

— JAR

Insert into the 11th paragraph, after the last list item:

— JAR

Insert after the 11th paragraph: The privileges for facilities defined in this document are as follows.

— The privileges required to invoke the `SQLJ.INSTALL_JAR`, `SQLJ.REPLACE_JAR`,and `SQLJ.REMOVE_JAR` procedures are implementation-defined (IW012).

  NOTE 5 — This is similar to the implementation-defined privileges required for creating a schema.

— Only the owner of the JAR is permitted to invoke the `SQLJ.ALTER_JAVA_PATH` procedure and the owner shall also have the USAGE privilege on each JAR referenced in the path argument.

— Invocations of <SQL-invoked routine> and <drop routine statement> to define and drop external Java routines are governed by the normal Access Rules for SQL-schema statements.

— Invocations of Java methods referenced by SQL names are governed by the normal EXECUTE privilege on SQL routine names.

  It is implementation-defined (IA134) whether a Java method called by an SQL name executes with “definer’s rights” or “invoker’s rights” — that is, whether it executes with the user-name of the user who performed the <SQL-invoked routine> or the user-name of the current user.

4.12 JARs

4.12.1 Introduction to JARs

A JAR is a Java archive containing a set of Java `class` and `ser` files and optionally a deployment descriptor file. Installed JARs provide the implementation of external Java routines and external Java data types to an SQL-environment.

JARs are created outside the SQL-environment. They are copied into the SQL-environment by the `SQLJ.INSTALL_JAR` procedure. No subsequent SQL statement or procedure modifies an installed JAR in any way, other than to remove it from the SQL-environment, to replace it in its entirety, or to alter its SQL-Java path. In particular, no SQL operation adds classes to a JAR, removes classes from a JAR, or replaces classes in a JAR. The reason for this “no modification” principle for installed JAR is that JARs are often signed, and often contain *manifest* data that might be invalidated by modification of JARs by the SQL-environment.

Each installed JAR is represented by a *JAR descriptor*. A JAR descriptor contains the following.

— The catalog name, schema name, and JAR identifier of the JAR.

— The SQL-Java path of the JAR.

4.12.2 Deployment descriptor files

When a JAR is installed, one or more <SQL-invoked routine>s that define external Java routines shall be executed before the static methods of its contained Java classes can be used as SQL-invoked routines, and one or more <user-defined type definition>s shall be executed before its contained classes can be used as user-defined types. In addition, <grant privilege statement>s may be required to define privileges for newly created SQL-invoked routines and user-defined types. Later, when a JAR is removed, corresponding <drop routine statement>s, <drop data type statement>s, and <revoke statement>s shall be executed.

If a JAR is to be installed in several SQL-implementations, the <SQL-invoked routine>s, <user-defined type definition>s, <user-defined ordering definition>s, <grant privilege statement>s, <drop routine statement>s, <drop data type statement>s, <drop user-defined ordering statement>s, and <revoke statement>s will often be the same for each SQL-implementation. To assist the automation of repeated installations, deployment descriptor files contain the variants of SQL-schema statements defined in this document. These statements are grouped into multi-statement *install actions* and *remove actions* respectively executed by `SQLJ.INSTALL_JAR` and `SQLJ.REMOVE_JAR` procedures when deployment is requested. In addition, an *implementor block* is provided to allow specification of custom install and remove actions. Since the SQL-schema statements refer to their containing JAR in the <SQL-invoked routine>s and <user-defined type definition>s, within a deployment descriptor file, the JAR name “`thisjar`” is used as a place holder JAR name for the containing JAR.

This document provides a mechanism to execute its variants of SQL-schema statements, namely by requesting deployment during invocation of `SQLJ.INSTALL_JAR` and `SQLJ.REMOVE_JAR` procedures. A conforming SQL-implementation is required to support either deployment descriptor based execution of its SQL-schema statements (Feature J531, “Deployment”) or another standard statement execution mechanism such as direct invocation or embedded SQL (Feature J511, “Commands”); a conforming SQL-implementation is not required to support both mechanisms.

5 Lexical elements

This Clause modifies Clause 5, "Lexical elements", in ISO/IEC 9075-2.

5.1 <token> and <separator>

This Subclause modifies Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

Function

Specify lexical units (tokens and separators) that participate in SQL language.

Format

```
<reserved word> ::=
    !! All alternatives from ISO/IEC 9075-2
  | JAR

<non-reserved word> ::=
    !! All alternatives from ISO/IEC 9075-2
  | COMPARABLE

  | INSTALL | INTERFACE

  | JAVA

  | REMOVE

  | SQLDATA

  | VARIABLE
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

5.2 Names and identifiers

This Subclause modifies Subclause 5.4, "Names and identifiers", in ISO/IEC 9075-2.

Function

Specify names.

Format

```
<jar name> ::=
  [ <schema name> <period> ] <jar id>

<jar id> ::=
  <identifier>

<Java class name> ::=
  [ <packages> <period> ] <class identifier>

<jar and class name> ::=
  <jar id> <colon> <Java class name>

<qualified Java field name> ::=
  [ <Java class name> <period> ] <Java field name>

<packages> ::=
  <package identifier> [ <period> <package identifier> ]...

<package identifier> ::=
  <Java identifier>

<class identifier> ::=
  <Java identifier>

<Java field name> ::=
  <Java identifier>

<Java method name> ::=
  <Java identifier>

<Java identifier> ::=
  !! See the Syntax Rules.
```

Syntax Rules

1) Insert after the last SR: <Java identifier> shall be a valid identifier according to the rules of Java parsing and lexical analysis.

   NOTE 6 — The rules of Java parsing and lexical analysis are specified in Java.

2) Insert after the last SR: The character set supported is implementation-defined (IV224), and the maximum lengths of the <package identifier>, <class identifier>, <Java field name>, and <Java method name> are implementation-defined (IL069).

3) Insert after SR 18): Two <jar name>s are equivalent if and only if they have equivalent <jar id>s and equivalent implicit or explicit <schema name>s.

Access Rules

No additional Access Rules.

General Rules

1) Insert after the last GR: A <jar name> identifies a JAR.
2) Insert after the last GR: A <jar id> represents an unqualified JAR name.
3) Insert after the last GR: A <Java class name> identifies a fully qualified Java class.
4) Insert after the last GR: A <packages> identifies a fully qualified Java package.
5) Insert after the last GR: A <package identifier> represents an unqualified Java package name.
6) Insert after the last GR: A <class identifier> represents an unqualified Java class name.
7) Insert after the last GR: A <Java field name> represents the name of a field within a Java class.
8) Insert after the last GR: A <Java method name> represents the name of a method within a Java class.

Conformance Rules

No additional Conformance Rules.

6 Scalar expressions

This Clause modifies Clause 6, "Scalar expressions", in ISO/IEC 9075-2.

6.1 <method invocation>

This Subclause modifies Subclause 6.18, "<method invocation>", in ISO/IEC 9075-2.

Function

Reference an SQL-invoked method of a user-defined type value.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 3): If *UDT* is an external Java data type, then <method invocation> shall immediately contain <direct invocation>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

6.2 <new specification>

This Subclause modifies Subclause 6.20, “<new specification>”, in ISO/IEC 9075-2.

Function

Invoke a method on a newly-constructed value of a structured type.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after the last GR: If Feature J571, “NEW operator” is not supported, then the mechanism used to invoke a constructor of an external Java data type is implementation-defined (IW202).

Conformance Rules

1) Insert after the last CR: Without Feature J571, “NEW operator”, conforming SQL language shall not contain a <new specification> in which the schema identified by the implicit or explicit <schema name> of the <routine name> *RN* immediately contained in <routine invocation> immediately contained in the <new specification> contains a user-defined type whose user-defined type name is *RN* and that is an external Java data type.

7 Predicates

This Clause modifies Clause 8, “Predicates”, in ISO/IEC 9075-2.

7.1 <comparison predicate>

This Subclause modifies Subclause 8.2, “<comparison predicate>”, in ISO/IEC 9075-2.

Function

Specify a comparison of two row values.

Format

No additional Format items.

Syntax Rules

1) Augment NOTE 331 by adding “If the comparison category is COMPARABLE, then no comparison functions are specified for *T1* and *T2*.” at the end of the note.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1)b)iv)3): If the comparison category of UDT_X is COMPARABLE, then:

   a) The subject SQL data type shall be an external Java data type. Let *JC* be the subject Java class of that external Java data type.

      NOTE 7 — Syntax Rules in Subclause 10.11, “<user-defined ordering definition>”, require that *JC* implement the Java interface `java.lang.Comparable`. The interface `java.lang.Comparable` requires an implementing Java class to have a method named `compareTo`, whose result data type is Java`int`.

   b) Let *XJV* be the value of *X* in the associated JVM. Let *YJV* be the value of *Y* in that associated JVM.

   c) `X = Y`

      has the same result as if the JVM executed the Java boolean expression

      `XJV.compareTo(YJV) == 0`

   d) `X < Y`

      has the same result as if the JVM executed the Java boolean expression

      `XJV.compareTo(YJV) < 0`

   e) `X <> Y`

has the same result as if the JVM executed the Java boolean expression

```
XJV.compareTo(YJV) != 0
```

f) `X > Y`

has the same result as if the JVM executed the Java boolean expression

```
XJV.compareTo(YJV) > 0
```

g) `X <= Y`

has the same result as if the JVM executed the Java boolean expression

```
XJV.compareTo(YJV) <= 0
```

h) `X >= Y`

has the same result as if the JVM executed the Java boolean expression

```
XJV.compareTo(YJV) >= 0
```

Conformance Rules

No additional Conformance Rules.

8 Additional common rules

This Clause modifies Clause 9, "Additional common rules", in ISO/IEC 9075-2.

8.1 Invoking an SQL-invoked routine

This Subclause modifies Subclause 9.18, "Invoking an SQL-invoked routine", in ISO/IEC 9075-2.

Function

Execute the invocation of an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

1) Insert after the last SR: If *SR* is an external Java routine, then:

   a) No <SQL argument> immediately contained in <SQL argument list> shall immediately contain <generalized expression>.

   b) If validation of the <Java parameter declaration list> has been implementation-defined (IA136) to be performed by <routine invocation>, then the Syntax Rules of Subclause 9.5, "Java routine signature determination", are applied with <routine invocation> as *ELEMENT*, 0 (zero) as *INDEX*, and *SR* as *SUBJECT*.

Access Rules

No additional Access Rules.

General Rules

NOTE 8 — The General Rules of this Subclause as defined in GR 2) of Subclause 9.18, "Invoking an SQL-invoked routine", in ISO/IEC 9075-2 are not always terminated when an exception condition is raised.

1) Insert after GR 4)c)ii)1): If *R* is an external Java routine, then let CPV_i be an implementation-defined (IV245) non-null value of declared type T_i.

2) Insert before GR 5): If *R* is an external Java routine that is not a static field method, then let *P* be the *subject Java method* of *R*.

   NOTE 9 — The subject Java method of an external Java routine is defined in Subclause 9.5, "Java routine signature determination".

3) Augment GR 5) by adding "*R* is not an external Java routine " as an additional restriction to the predicate in the lead text of the Rule.

4) Augment GR 6)f)ii) by adding "*R* is not a static field method " as a restriction to the lead text of the Rule.

5) Insert after GR 6)f)ii): Otherwise, the following are copied from *RSC* to *CSC*:

   a) The identities of all temporary tables.

   b) The cursor instance descriptors of every open cursor.

   c) All prepared statements.

   d) All SQL descriptor areas.

   e) Every currently available result set sequence *RSS*, along with the specific name of an SQL-invoked procedure *SIP* and the name of the invoker of *SIP* for the invocation causing *RSS* to be brought into existence.

6) Insert before GR 9)d): If *R* specifies PARAMETER STYLE JAVA, then

   Case:

   a) If *R* is an SQL-invoked function that is an array-returning external function or a multiset-returning external function, then the effective SQL parameter list *ESPL* of *R* is set as follows:

      i) If *R*'s returned array's element type or returned multiset's element type is a row type, then let *FRN* be the degree of the element type; otherwise, let *FRN* be 1 (one).

      ii) For *i* ranging from 1 (one) to *PN*, the *i*-th entry in *ESPL* is set to CPV_i.

      iii) For *i* ranging from *PN*+1 to *PN*+*FRN*, the *i*-th entries in *ESPL* are the *result data items*.

      iv) For *i* equal to *PN*+*FRN*+1, the *i*-th entry in *ESPL* is the save area data item and for *i* equal to *PN*+*FRN*+2, the *i*-th entry in *ESPL* is the *call type data item*.

      v) Set the value of the save area data item (that is, SQL argument value list entry *PN*+*FRN*+1) to null and set the value of the call type data item (that is, SQL argument value list entry *PN*+*FRN*+2) to −1 (negative one).

      NOTE 10 — Initialization of the save area data item occurs in Subclause 8.2, "Execution of array-returning functions"; for now, it is set to null.

   b) Otherwise, for *i* ranging from 1 (one) to *PN*, let the effective SQL parameter list *ESPL* of *R* be the list of values CPV_i.

7) Augment GR 9)g)ii)1) by adding "*R* is not an external Java routine " as an additional restriction to the predicate in the lead text of the Rule.

8) Insert before GR 9)g)ii)3): If *R* is an external Java routine and *R* is not a collection-returning external function, then *P* is executed in a manner determined as follows and with a list of parameters PD_i whose values are set as follows:

   a) Let *SRD* be routine descriptor of *R*.

   b) If *SRD* indicates that *R* is an SQL-invoked method, then let *SRUDT* be the user-defined type whose descriptor contains *SR*'s corresponding method specification descriptor *MSD* and let *JCLSN* be the subject Java class of *SRUDT*.

   c) Case:

      i) If *SRD* indicates that *R* is an SQL-invoked method and *MSD* indicates that *R* is a static field method, then:

         1) Let *JSF* be the subject static field of *R*.

NOTE 11 — The “subject static field” of an SQL-invoked method is defined in Subclause 9.5, “Java routine signature determination”.

2) Let *ERT* be the effective returns data type of *R*.

NOTE 12 — “effective returns data type” is defined in the Syntax Rules of Subclause 9.18, “Invoking an SQL-invoked routine”, in ISO/IEC 9075-2.

3) Case:

A) If *ERT* is a user-defined type, then:

I) Let *SJCE* be the most specific Java class of the value of *JSF*, and let *STU* be the user-defined type whose subject Java class is *SJCE* and whose user-defined type is *ERT* or is a subclass of *ERT*.

II) Let *UIS* be the <interface specification> specified by the user-defined type descriptor of *STU*.

Case:

1) If *UIS* is SERIALIZABLE, then:

a) The subject Java class *SJCE*’s `writeObject()` method is executed to convert the Java value of *JSF* to the SQL value *SSFV* of user-defined type *STU*.

b) The method of execution of the subject Java class’s implementation of `writeObject()` is implementation-defined (IW170).

NOTE 13 — If *UIS* is SERIALIZABLE, then, as described in Subclause 10.4, “<user-defined type definition>”, the descriptor’s subject Java class implements the Java interface `java.io.Serializable` and defines that interface’s `writeObject()` method as described by the JVMS.

2) If *UIS* is SQLDATA, then:

a) The subject Java class *SJCE*’s method `writeSQL()` is executed to convert the Java value of *JSF* to the SQL value *SSFV* of user-defined type *STU*.

b) The method of execution of the subject Java class’s implementation of `writeSQL()` is implementation-defined (IW171).

NOTE 14 — If *UIS* is SQLDATA, then, as described in Subclause 10.4, “<user-defined type definition>”, the descriptor’s subject Java class implements the Java interface `java.sql.SQLData` and defines that interface’s `writeSQL()` method as described by JDBC and JVMS.

B) Otherwise, the value of *SSFV* is set to the value of *JSF*.

4) Let *RESULT* be an arbitrary site of declared type *ERT*. The General Rules of Subclause 9.2, “Store assignment”, in ISO/IEC 9075-2, are applied with *SSFV* as *VALUE* and *RESULT* as *TARGET*. The result of the <routine invocation> is the value of *RESULT*. No further General Rules of this Subclause are applied.

ii) Otherwise:

1) Let *JPDL* be an ordered list of the data types of the Java parameters declared for *P* in the order they appear in *P*’s declaration.

NOTE 15 — If any Java parameter is declared to be of an array class, then *JPDL* reflects that information.

2) If *SRD* indicates that *R* is an SQL-invoked method and *MSD* indicates that *R* is an instance method or a constructor method, then prefix *JPDL* with the subject parameter as follows.

Case:

A) If *JPDL* contains one or more Java data types, then prefix *JPDL* with *JCLSN*.

B) Otherwise, replace *JPDL* with *JCLSN*.

3) Let JP_i be the *i*-th data type in *JPDL*.

4) For *i* ranging from 1 (one) to *EN*, if JP_i is of an array class, then let JP_i be the component type of JP_i.

NOTE 16 — The component type of a Java array is defined in Java.

5) For *i* ranging from 1 (one) to *EN*, if ESP_i is the SQL null value and if JP_i is any of `boolean`, `byte`, `short`, `int`, `long`, `float`, or `double`, then an exception condition is raised: *external routine invocation exception — null value not allowed (39004)*.

6) For i ranging from 1 (one) to EN,

Case:

A) If the declared type of ESP_i is a user-defined type, then let the most specific type of ESP_i be *U*, let *UIS* be the <interface specification> specified by the user-defined type descriptor of *U*, and let *SJCU* be the subject Java class of *U*.

Case:

I) If *UIS* is SERIALIZABLE, then:

1) The subject Java class *SJCU*'s method `readObject()` is executed to convert the value of ESP_i to a Java object, the value of PD_i.

2) The method of execution of the subject Java class's implementation of `readObject()` is implementation-defined (IW172).

NOTE 17 — If *UIS* is SERIALIZABLE, then, as described in Subclause 10.4, "<user-defined type definition>", the subject Java class of *U* implements the Java interface `java.io.Serializable` and defines that interface's `readObject()` method as described by JVMS.

II) If *UIS* is SQLDATA, then:

1) The subject Java class *SJCU*'s method `readSQL()` is executed to convert the value of ESP_i to a Java object, the value of PD_i.

2) The method of execution of the subject Java class's implementation of `readSQL()` is implementation-defined (IW173).

NOTE 18 — If *UIS* is SQLDATA, then, as described in Subclause 10.4, "<user-defined type definition>", the subject Java class of *U* implements the Java interface `java.sql.SQLData` and defines that interface's `readSQL()` method as described by JDBC and JVMS.

B) Otherwise, the value of PD_i, of the Java data type JP_i, is set to the value of ESP_i.

7) For *i* ranging from 1 (one) to *EN*, if P_i is an output SQL parameter or both an input SQL parameter and an output SQL parameter, then:

   A) Let PAD_i be a Java array of length 1 (one) and data type JP_i initialized as specified in Java.

      NOTE 19 — PAD_i is a Java object effectively created by execution of the Java expression new JP_i[1].

   B) If P_i is both an input SQL parameter and an output SQL parameter, then PAD_i[0] is set to PD_i.

   C) PD_i is replaced by PAD_i.

8) Let *JPEN* be the number of Java data types in *JPDL*.

9) If *JPEN* is greater than *EN*, then prepare the Java parameters for the DYNAMIC RESULT SET parameters as follows.

   For *i* ranging from *EN*+1 to *JPEN*:

   A) Let PAD_i be a Java array of length 1 (one) and data type JP_i initialized as specified in Java.

      NOTE 20 — PAD_i is a Java object effectively created by execution of the Java expression new JP_i[1].

   B) The value of PD_i is set to the value of PAD_i.

10) Let *JCLSN*, *JMN*, and *ERT* be respectively the subject Java class name, the subject Java method name, and the effective returns data type of *R*. The subject Java method of the subject Java class is invoked as follows.

    Case:

    A) If *R* is an SQL-invoked procedure, then:

       I) If *JPEN* is greater than 0 (zero), then the following Java statement is effectively executed:

```
JCLSN.JMN ( PD1, ..., PDJPEN ) ;
```

       II) If *JPEN* equals 0 (zero), then the following Java statement is effectively executed:

```
JCLSN.JMN ( ) ;
```

    B) If *R* is an SQL-invoked method whose routine descriptor specifies STATIC or *R* is an SQL-invoked regular function, then:

       I) If *ERT* is a user-defined type, then let *SJCE* and *SJCEN* be the subject Java class and the subject Java class name of *ERT*, respectively.

       II) If *ERT* is not a user-defined type, then let *SJCEN* be the Java returns data type of the subject Java method.

       III) If *JPEN* is greater than 0 (zero), then the following Java statement is effectively executed:

```
SJCEN tempU =
JCLSN.JMN ( PD1, ..., PDJPEN ) ;
```

IV) If *JPEN* equals 0 (zero), then the following Java statement is effectively executed:

```
SJCEN tempU =
JCLSN.JMN ( ) ;
```

C) If *R* is an SQL-invoked constructor method, then:

I) If *JPEN* is greater than 1 (one), then the following Java statement is effectively executed:

```
JCLSN PD_1 = new
JCLSN ( PD_2, ...,
PD_JPEN ) ;
```

II) If *JPEN* equals 1 (one), then the following Java statement is effectively executed:

```
JCLSN PD_1 = new JCLSN ( ) ;
```

D) Otherwise:

I) If *ERT* is a user-defined type, then let *SJCE* and *SJCEN* be the subject Java class and the subject Java class name of *ERT*, respectively.

II) If *ERT* is not a user-defined type, then let *SJCEN* be the Java returns data type of the subject Java method.

III) If *JPEN* is greater than 1 (one), then the following Java statement is effectively executed:

```
SJCEN tempU =
    PD_1.JMN ( PD_2, ..., PD_JPEN ) ;
```

IV) If *JPEN* equals 1 (one), then the following Java statement is effectively executed:

```
SJCEN tempU = PD_1.JMN ( ) ;
```

NOTE 21 — The Java method effectively executed by either the Java statement `SJCEN tempU = PD_1.JMN ( PD_2, ..., PD_JPEN ) ;` or the Java statement `SJCEN tempU = PD_1.JMN ( ) ;` is determined based on the value of PD_1 according to Java's rules for overriding by instance methods, as specified in Java.

9) Insert after GR 9)g)ii)5): If *R* is an external Java routine, then the scope and persistence of any modifications of class variables made before the completion of any execution of *P* is implementation-dependent (UA006).

10) Insert before GR 9)h)i): If *R* is an external Java routine and the execution of *P* completes with an uncaught Java exception *E*, then let *EM* be the result of the Java method call `E.getMessage()`

a) Case:

i) If *E* is an instance of `java.sql.SQLException`, and the result *SS* of the Java method call `E.getSQLState()` is a five-character string, then let *C* be the first and second characters of *SS*, and let *SC* be the third, fourth, and fifth characters of *SS*.

ii) Otherwise, let *C* be '38' (corresponding to external routine exception) and *SC* be '000' (corresponding to no subclass).

b) An exception condition is raised with class *C*, subclass *SC*, and the associated message text *EM*.

11) Augment GR 9)h)ii) by adding "If *R* is not an external Java routine " as a restriction to the lead text of the Rule.

12) Insert after GR 9)i)i)3): If *R* is an external Java routine that is not a type-preserving function, then let *ERT* be the effective returns data type of *R*. The returned value of *P*, *tempU*, is processed as follows:

a) Case:

i) If *ERT* is a user-defined type, then:

1) Let *SJCE* be the most specific Java class of the value of *tempU*, and let *STU* be the user-defined type whose subject Java class is *SJCE* and whose user-defined type is *ERT* or is a subclass of *ERT*.

2) Let *UIS* be the <interface specification> specified by the user-defined type descriptor of *STU*.

3) Case:

A) If *UIS* is SERIALIZABLE, then:

I) The subject Java class *SJCE*'s method `writeObject()` is executed to convert the Java value of *tempU* to the SQL value *SSFV* of user-defined type *STU*.

II) The method of execution of the subject Java class's implementation of `writeObject()` is implementation-defined (IW170).

NOTE 22 — If *UIS* is SERIALIZABLE, then, as described in Subclause 10.4, "<user-defined type definition>", the descriptor's subject Java class implements the Java interface `java.io.Serializable` and defines that interface's `writeObject()` method as described by JVMS.

B) If *UIS* is SQLDATA, then:

I) The subject Java class *SJCE*'s method `writeSQL()` is executed to convert the Java value of *tempU* to the SQL value *SSFV* of user-defined type *STU*.

II) The method of execution of the subject Java class's implementation of `writeSQL()` is implementation-defined (IW171).

NOTE 23 — If *UIS* is SQLDATA, then as described in Subclause 10.4, "<user-defined type definition>", the descriptor's subject Java class implements the Java interface `java.sql.SQLData` and defines that interface's `writeSQL()` method as described by JDBC and JVMS.

ii) Otherwise, the value of *SSFV* is set to the value of *tempU*.

b) Let *RV* be *SSFV*.

13) Insert after GR 9)i)i)3): If *R* is an external Java routine that is a type-preserving function, then let *ERT* be the effective returns data type of *R*. The returned value of *P*, PD_1, is processed as follows:

a) Let *SJCE* be the most specific Java class of the value of PD_1, and let *STU* be the user-defined type whose subject Java class is *SJCE* and whose user-defined type is *ERT* or is a subclass of *ERT*.

b) Let *UIS* be the <interface specification> specified by the user-defined type descriptor of *STU*.

Case:

i) If *UIS* is SERIALIZABLE, then:

1) The subject Java class *SJCE*'s method `writeObject()` is executed to convert the Java value of PD_1 to the SQL value *SSFV* of user-defined type *STU*.

2) The method of execution of the subject Java class's implementation of `writeObject()` is implementation-defined (IW170).

NOTE 24 — If *UIS* is SERIALIZABLE, then as described in Subclause 10.4, "<user-defined type definition>", the descriptor's subject Java class implements the Java interface `java.io.Serializable` and defines that interface's `writeObject()` method as described by JVMS.

ii) If *UIS* is SQLDATA, then:

1) The subject Java class *SJCE*'s method `writeSQL()` is executed to convert the Java value of PD_1 to the SQL value *SSFV* of user-defined type *STU*.

2) The method of execution of the subject Java class's implementation of `writeSQL()` is implementation-defined (IW171).

NOTE 25 — If *UIS* is SQLDATA, then as described in Subclause 10.4, "<user-defined type definition>", the descriptor's subject Java class implements the Java interface `java.sql.SQLData` and defines that interface's `writeSQL()` method as described by JDBC and JVMS.

c) Let *RV* be *SSFV*.

14) Insert after GR 9)j)ii): If *R* specifies PARAMETER STYLE JAVA, then each parameter that is either an output SQL parameter or both an input SQL parameter and an output SQL parameter is processed as follows:

a) Let P_i be the *i*-th SQL parameter of *R* and let T_i be the declared type of P_i.

b) EPV_i is set to the value of PD_i[0].

Case:

i) If T_i is a user-defined type, then:

1) Let *SJCE* be the most specific Java class of the value of EPV_i, and let *STU* be the user-defined type whose subject Java class is *SJCE* and whose user-defined type is T_i or is a subclass of T_i.

2) Let *UIS* be the <interface specification> specified by the user-defined type descriptor of *STU*.

Case:

A) If *UIS* is SERIALIZABLE, then:

I) The subject Java class *SJCE*'s method `writeObject()` is executed to convert the Java value of EPV_i to the SQL value CPV_i of the user-defined type *STU*.

II) The method of execution of the subject Java class's implementation of `writeObject()` is implementation-defined (IW170).

NOTE 26 — If *UIS* is SERIALIZABLE, then as described in Subclause 10.4, "<user-defined type definition>", the descriptor's subject Java class implements the Java interface `java.io.Serializable` and defines that interface's `writeObject()` method as described by JVMS.

B) If *UIS* is SQLDATA, then:

I) The subject Java class *SJCE*'s method `writeSQL()` is executed to convert the Java value of EPV_i to the SQL value CPV_i of user-defined type *STU*.

II) The method of execution of the subject Java class's implementation of `writeSQL()` is implementation-defined (IW171).

NOTE 27 — If *UIS* is SQLDATA, then as described in Subclause 10.4, "<user-defined type definition>", the descriptor's subject Java class implements the Java interface `java.sql.SQLData` and defines that interface's `writeSQL()` method as described by JDBC and JVMS.

ii) Otherwise, CPV_i is set to EPV_i.

15) Convert GR 11)b) to be: Case:

a) If *R* is an external Java routine, then let *RSN* be a set containing the first element of each of the *JPEN*–*EN* arrays generated above for result set mappable parameters, let *RS* be the elements of *RSN* that are not equal to the Java null value, and let *OPN* be the number of elements in *RS*.

b) Otherwise, the original GR 11)b)

16) Insert before GR 11)d): If *R* is an external Java routine, then:

a) If the JDBC connection object that created any element of *RS* is closed, then the effect is implementation-defined (IA141).

b) If any element of *RS* is not an object returned by a connection to the current SQL system and SQL session, then the effect is implementation-defined (IA142).

17) Convert GR 11)d) to be: Case:

a) If *R* is an external Java routine, then let *FRC* be a copy of the elements of *RS* that remain open in the order that they were opened in SQL. Let FRC_i, 1 (one) $\leq i \leq RTN$, be the *i*-th cursor in *FRC*, let RCS_i be the result set of FRC_i.

b) Otherwise, the original GR 11)d)

18) Convert GR 11)f) to be: Case:

a) If *R* is an external Java routine, then for each result set RS_i in *RS*, close RS_i and close the statement object that created RS_i.

b) Otherwise, the original GR 11)f)

19) Insert before GR 14): If *R* is an external Java routine, then whether the call of *P* returns update counts as defined in JDBC is implementation-defined (IA143).

Conformance Rules

No additional Conformance Rules.

8.2 Execution of array-returning functions

This Subclause modifies Subclause 9.21, "Execution of array-returning external functions", in ISO/IEC 9075-2.

Function

Define the execution of an external function that returns an array value.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Convert GR 6) to be: Case:

   a) If *P* is an external Java routine then let *PN* and *N* be *EN*−*FRN*−2.

   b) Otherwise, the original GR 6)

2) Augment GR 7) by adding "If *P* is not an external Java routine " as a restriction to the lead text of the Rule.

3) Augment GR 9) by adding "*P* is not an external Java routine " as an additional restriction to the predicate in the lead text of the Rule.

4) Insert before GR 10): If *P* is an external Java routine, and the call type data item has a value of −1 (negative one, indicating *open call*), then *P* is executed with a list of parameters PD_i, 1 (one) ≤ *i* ≤ *EN*, whose values are set as follows:

   a) Let *JPDL* be an ordered list of the data types of the Java parameters declared for *P* in the order they appear in *P*'s declaration.

      NOTE 28 — If any Java parameter is declared to be of an array class, then *JPDL* reflects that information.

   b) Let $JPDT_i$ be the *i*-th Java data type in *JPDL*.

   c) For *i* ranging from *PN*+1 to *PN*+*FRN*, let JP_i be the component type of $JPDT_i$.

      NOTE 29 — The component type of a Java array is defined in Java.

   d) For *i* ranging from 1 (one) to *PN*, if the value of ESP_i is the SQL null value and if $JPDT_i$ is any of `boolean`, `byte`, `short`, `int`, `long`, `float`, or `double`, then an exception condition is raised: *external routine invocation exception — null value not allowed (39004)*.

   e) For *i* ranging from 1 (one) to *PN*,

      Case:

      i) If ESP_i is a user-defined type, then let the most specific type of ESP_i be *U*, let *UIS* be the <interface specification> specified by the user-defined type descriptor of *U*, and let *SJCU* be the subject Java class of *U*.

Case:

1) If *UIS* is SERIALIZABLE, then:

   A) *SJCU*'s method `readObject()` is executed to convert the value of ESP_i to a Java object, the value of PD_i.

   B) The method of execution of the subject Java class's implementation of `readObject()` is implementation-defined (IW172).

   NOTE 30 — If *UIS* is SERIALIZABLE, then, as described in Subclause 10.4, "<user-defined type definition>", the subject Java class of *U* implements the Java interface `java.io.Serializable` and defines that interface's `readObject()` method as described by JVMS.

2) If *UIS* is SQLDATA, then:

   A) *SJCU*'s method `readSQL()` is executed to convert the value of ESP_i to a Java object, the value of PD_i.

   B) The method of execution of the subject Java class's implementation of `readSQL()` is implementation-defined (IW173).

   NOTE 31 — If *UIS* is SQLDATA, then, as described in Subclause 10.4, "<user-defined type definition>", the subject Java class of *U* implements the Java interface `java.sql.SQLData` and defines that interface's `readSQL()` method as described by JDBC and JVMS.

ii) Otherwise, the value of PD_i is set to the value of ESP_i .

f) For *i* ranging from *PN*+1 to *PN*+*FRN*:

i) Let PAD_i be a Java array of length 1 (one) and data type JP_i initialized as specified in Java.

NOTE 32 — PAD_i is a Java object effectively created by execution of the Java expression new JP_i[1].

ii) PD_i is replaced by PAD_i.

g) For the save area data item, for *i* equal to *EN*−1:

i) Case:

1) If the Java data type $JPDT_i$ is an array class of `java.lang.String`, then let PAD_i be a Java array of length 1 (one) of `java.lang.String`, initialized as specified in Java.

   NOTE 33 — PAD_i is a Java object effectively created by execution of the Java expression `new java.lang.String[1]`.

2) Otherwise, create a `java.lang.StringBuffer` of the implementation-defined (IV196) length of a save area data item. Let *LN* be that implementation-defined length and let PAD_i be the Java object effectively created by execution of the Java expression `new java.lang.StringBuffer(LN)`. Then initialize PAD_i with *LN null characters* (U+0000).

ii) PD_i is replaced by PAD_i .

h) For the *call type data item*, for *i* equal to *EN*, the value of PD_i is set to the value −1 (negative one, indicating *open call*).

i) Let *JCLSN* and *JMN* be respectively the subject Java class name, and the subject Java method name of *P*. The following Java statement is effectively executed:

```
    JCLSN.JMN( PD_1, ..., PD_EN );
```

5) Augment GR 10)a) by adding "*P* is not an external Java routine " as an additional restriction to the predicate of the Rule.

6) Augment GR 10)b) by adding "*P* is not an external Java routine " as an additional restriction to the predicate of the Rule.

7) Insert before GR 10)c): If *P* is an external Java routine and the prior invocation of *P* did not terminate with an unhandled Java exception, then set the call type data item to 0 (zero) (indicating *fetch call*).

8) Augment GR 10)c) by adding "*P* is not an external Java routine " as an additional restriction to the predicate in the lead text of the Rule.

9) Insert before GR 10)d): If *P* is an external Java routine and the prior invocation of *P* terminated with an unhandled Java exception that is an instance of the class `java.sql.SQLException`, or a subclass of such a class, and the result of invoking the method `getSQLState()` against that instance is a `java.lang.String` whose value is '02000' (corresponding to the completion condition *no data (02000)*) then:

   a) If each JP_i for *i* ranging from *PN*+1 to *PN*+*FRN* that is a Java class has an associated value in the first element, ([0]), of PD_i that is a Java null, then set *AR* to the null value.

   b) Set the call type data item to 1 (one) (indicating *close call*).

10) Augment GR 11)a) by adding "IF *P* is not an external Java routine " as a restriction to the lead text of the Rule.

11) Insert before GR 11)d): If *P* is an external Java routine, then *P* is executed with a list of *EN* parameters PD_i and whose values are set as follows:

   a) For *i* ranging from 1 (one) to *PN*,

      Case:

      i) If ESP_i is a user-defined type, then let the most specific type of ESP_i be *U*, let *UIS* be the <interface specification> specified by the user-defined type descriptor of *U*, and let *SJCU* be the subject Java class of *U*.

         Case:

         1) If *UIS* is SERIALIZABLE, then:

            A) *SJCU*'s method `readObject()` is executed to convert the value of ESP_i to a Java object, the value of PD_i .

            B) The method of execution of the subject Java class's implementation of `readObject()` is implementation-defined (IW172).

            NOTE 34 — If UIS is SERIALIZABLE, then, as described in Subclause 10.4, "<user-defined type definition>", the subject Java class of *U* implements the Java interface `java.io.Serializable` and defines that interface's `readObject()` method as described by JVMS.

         2) If *UIS* is SQLDATA, then:

            A) *SJCU*'s method `readSQL()` is executed to convert the value of ESP_i to a Java object, the value of PD_i .

            B) The method of execution of the subject Java class's implementation of `readSQL()` is implementation-defined (IW173).

NOTE 35 — If *UIS* is SQLDATA, then, as described in Subclause 10.4, "<user-defined type definition>", the subject Java class of *U* implements the Java interface `java.sql.SQLData` and defines that interface's `readSQL()` method as described by JDBC and JVMS.

ii) Otherwise, the value of PD_i is set to the value of ESP_i.

b) For *i* ranging from *PN*+1 to *PN*+*FRN*:

i) Let PAD_i be a Java array of length 1 (one) and data type JP_i initialized as specified in Java.

NOTE 36 — PAD_i is a Java object effectively created by execution of the Java expression new JP_i [1].

ii) PD_i is replaced by PAD_i .

c) For the save area data item, for *i* equal to *EN*−1:

i) Case:

1) If the Java data type JP_i is an array class of `java.lang.String`, then let PAD_i be a Java array of length 1 (one) of `java.lang.String`, containing the value of the `java.lang.String` returned by the prior execution of *P*.

2) Otherwise, let PAD_i be a `java.lang.StringBuffer` of length *LN* containing the value of the `java.lang.StringBuffer` returned by the prior execution of *P*.

ii) PD_i is replaced by PAD_i.

d) For the call type data item, for *i* equal to *EN*, the value of PD_i is set to the value 0 (zero) (indicating *fetch call*).

e) Let *JCLSN* and *JMN* be respectively the subject Java class name, and the subject Java method name of *P*. The following Java statement is effectively executed:

$JCLSN.JMN(\ PD_1,\ \ldots,\ PD_{EN}\ );$

12) Augment GR 11)d)i) by adding "*P* is not an external Java routine" as additional predicate in lead text of the rule.

13) Augment GR 11)d)i)3)A) by adding "*P* is not an external Java routine " as an additional restriction to the predicate of the Rule.

14) Insert after GR 11)d)i)3)A): If *P* is an external Java routine and each JP_i for *i* ranging from *PN*+1 to *PN*+*FRN* that is a Java class has an associated value of the first element, ([0]), of PD_i that is a Java null, then let the *E*-th element of *AR* be the null value.

15) Augment GR 11)d)i)3)B)I) by adding "*P* is not an external Java routine " as an additional restriction to the predicate of the Rule.

16) Insert after GR 11)d)i)3)B)I): If *P* is an external Java routine, then for the result data items, for i ranging from *PN*+1 through *PN*+*FRN*:

a) Case:

i) If ESP_i is a user-defined type, then:

1) Let EST_i be the most specific type of the value of ESP_i.

2) Let *SJCE* be the most specific Java class of the value of PD_i [0], and let *STU* be the user-defined type whose subject Java class is *SJCE* and whose user-defined type is EST_i or is a subclass of EST_i.

3) Let *UIS* be the <interface specification> specified by the user-defined type descriptor of *STU*.

4) Case:

A) If *UIS* is SERIALIZABLE, then:

I) *SJCE*'s method `writeObject()` is executed to convert the value of PD_i [0] to the value SC_i of user-defined type *STU*.

II) The method of execution of the subject Java class's implementation of `writeObject()` is implementation-defined (IW170).

NOTE 37 — If *UIS* is SERIALIZABLE, then, as described in Subclause 10.4, "<user-defined type definition>", the descriptor's subject Java class implements the Java interface `java.io.Serializable` and defines that interface's `writeObject()` method as described by JVMS.

B) If *UIS* is SQLDATA, then:

I) *SJCE*'s method `writeSQL()` is executed to convert the value of PD_i [0] to the value SC_i of user-defined type *STU*.

II) The method of execution of the subject Java class's implementation of `writeSQL()` is implementation-defined (IW171).

NOTE 38 — If *UIS* is SQLDATA, then as described in Subclause 10.4, "<user-defined type definition>", the descriptor's subject Java class implements the Java interface `java.sql.SQLData` and defines that interface's `writeSQL()` method as described by JDBC and JVMS.

ii) Otherwise, the value of SC_i is set to the value of PD_i [0].

b) Case:

i) If *FRN* is 1 (one), then let the *E*-th element of *AR* be SC_i.

ii) Otherwise, let the *E*-th element of *AR* be the value of the following <row value expression>:

```
ROW ( SV1, ..., SVFRN )
```

17) Augment GR 11)d)i) by adding "If *P* is not an external Java routine " as an additional restriction at the beginning of the Rule.

18) Augment GR 11)d)ii) by adding "*P* is an external Java routine and the prior invocation of *P* terminated with an unhandled Java exception that is an instance of the class `java.sql.SQLException`, or a subclass of such a class, and the result of invoking the method `getSQLState()` against that instance is a `java.lang.String` whose value is '02000' (corresponding to the completion condition *no data (02000)*) " as an alternative restriction for the predicate at the beginning of the Rule.

19) Augment GR 12) by adding "*P* is not an external Java routine " as an additional restriction to the predicate in the lead text of the Rule.

20) Insert after GR 12): If *P* is an external Java routine and the call type data item has a value of 1 (one) (indicating close call), then *P* is executed with a list of *EN* parameters PD_i and whose values are set as follows:

a) For *i* ranging from 1 (one) to *PN*,

Case:

i) If ESP_i is a user-defined type, then let the most specific type of ESP_i be *U*, let *UIS* be the <interface specification> specified by the user-defined type descriptor of *U*, and let *SJCU* be the subject Java class of *U*.

Case:

1) If *UIS* is SERIALIZABLE, then:

A) *SJCU*'s method `readObject()` is executed to convert the value of ESP_i to a Java object, the value of PD_i .

B) The method of execution of the subject Java class's implementation of `readObject()` is implementation-defined (IW172).

NOTE 39 — If *UIS* is SERIALIZABLE, then, as described in Subclause 10.4, "<user-defined type definition>", the subject Java class of *U* implements the Java interface `java.io.Serializable` and defines that interface's `readObject()` method as described by JVMS.

2) If *UIS* is SQLDATA, then:

A) *SJCU*'s method `readSQL()` is executed to convert the value of ESP_i to a Java object, the value of PD_i .

B) The method of execution of the subject Java class's implementation of `readSQL()` is implementation-defined (IW173).

NOTE 40 — If *UIS* is SQLDATA, then, as described in Subclause 10.4, "<user-defined type definition>", the subject Java class of *U* implements the Java interface `java.sql.SQLData` and defines that interface's `readSQL()` method as described by JDBC and JVMS.

ii) Otherwise, the value of PD_i is set to the value of ESP_i .

b) For *i* ranging from *PN*+1 to *PN*+*FRN*:

i) Let PAD_i be a Java array of length 1 (one) and data type JP_i initialized as specified in Java.

NOTE 41 — PAD_i is a Java object effectively created by execution of the Java expression `new` JP_i `[1]`.

ii) PD_i is replaced by PAD_i .

c) For the save area data item, for *i* equal to *EN*−1:

i) Case:

1) If the Java data type JP_i is an array class of `java.lang.String`, then let PAD_i be a Java array of length 1 (one) of `java.lang.String`, containing the value of the `java.lang.String` returned by the prior execution of *P*.

2) Otherwise, let PAD_i be a `java.lang.StringBuffer` of length *LN* containing the value of the `java.lang.StringBuffer` returned by the prior execution of *P*.

ii) PD_i is replaced by PAD_i.

d) For the call type data item, for *i* equal to *EN*, the value of PD_i is set to the value 1 (one) (indicating *close call*).

e) Let *JCLSN* and *JMN* be respectively the subject Java class name, and the subject Java method name of *P*. The following Java statement is effectively executed:

```
JCLSN.JMN( PD_1, ..., PD_EN );
```

Conformance Rules

No additional Conformance Rules.

9 Additional common elements

This Clause modifies Clause 10, "Additional common elements", in ISO/IEC 9075-2.

9.1 <language clause>

This Subclause modifies Subclause 10.2, "<language clause>", in ISO/IEC 9075-2.

Function

Specify a programming language.

Format

```
<language name> ::=
    !! All alternatives from ISO/IEC 9075-2
  | JAVA
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

Insert into Table 20, "Standard programming languages" the rows of Table 1, "Standard programming languages".

Table 1 — Standard programming languages

Language keyword	Relevant standard
JAVA	Java

Conformance Rules

No additional Conformance Rules.

9.2 <Java parameter declaration list>

Function

Specify the Java types of parameters for a Java method.

Format

```
<Java parameter declaration list> ::=
  <left paren> [ <Java parameters> ] <right paren>

<Java parameters> ::=
  <Java data type> [ { <comma> <Java data type> }... ]

<Java data type> ::=
  !! See the Syntax Rules.
```

Syntax Rules

1) A <Java data type> is a Java data type that is mappable or result set mappable, as specified in Subclause 4.6, "Parameter mapping". The <Java data type> names are case sensitive, and shall be fully qualified with their package names, if any.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature J631, "Java signatures", conforming SQL language shall not contain a <Java parameter declaration list> that is not equivalent to the default Java method signature as determined in Subclause 9.5, "Java routine signature determination".

9.3 <SQL Java path>

Function

Control the resolution of Java classes across installed JARs.

Format

```
<SQL Java path> ::=
  [ <path element>... ]

<path element> ::=
  <left paren> <referenced class> <comma> <resolution jar> <right paren>

<referenced class> ::=
    [ <packages> <period> ] <asterisk>
  | [ <packages> <period> ] <class identifier>

<resolution jar> ::=
  <jar name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) When a Java class *CJ* in a JAR *J* is executed in an SQL-implementation, let *P* be the SQL-Java path from *J*'s JAR descriptor.

NOTE 42 — A JAR descriptor's SQL-Java path is set by an invocation of the `SQLJ.ALTER_JAVA_PATH` procedure.

2) No `Class-Path` attribute affects class resolution. Every static or dynamic reference in *CJ* to a class with the name *CN* that is not a system class and is not contained in *J* is resolved as follows.

For each <path element> *PE* (if any) in *P*, in the order in which they were specified:

a) Let *RC* and *RJ* be the <referenced class> and <resolution jar>, respectively, contained in *PE*. Let *JR* be the JAR referenced by *RJ*.

b) If *RJ* is not the name of an installed JAR, then an exception condition is raised: *Java-related condition — invalid JAR name in path (46102)*.

NOTE 43 — This exception can only occur if the implementation-defined (IA153) action taken for an `SQLJ.ALTER_JAVA_PATH` call that raised an exception results in leaving invalid <jar name>s in the SQL-Java path.

c) If *RC* is equivalent to *CN*, then:

i) If *CN* is the name of some class *C* in *JR*, then *CN* resolves to class *C*.

ii) If *CN* is not the name of a class in *JR*, then an exception condition is raised: *Java-related condition — unresolved class name (46103)*.

d) If *RC* simply contains <asterisk>and simply contains <packages>, then let *PKG* be the specified <packages> and let *CI* be the <class identifier> of *CN*. If the <Java class name> of *CN* is *PKG.CI*, then:

i) If *CN* is the name of a class *C* in *JR*, then *CN* resolves to class *C*.

ii) If *CN* is not the name of a class in *JR*, then an exception condition is raised: *Java-related condition — unresolved class name (46103)*.

e) If *RC* simply contains <asterisk> and does not simply contain <packages>, then:

i) If *CN* is the name of a class *C* in *JR*, then *CN* resolves to class *C*.

ii) If *CN* is not the name of a class in *RJ*, then *CN* is not resolved by the <path element> being considered and the next <path element> in *P* is considered.

3) If *CN* is not resolved after all <path element>s in *P* have been considered, then an exception condition is raised: *Java-related condition — unresolved class name (46103)*.

Conformance Rules

1) Without Feature J601, "SQL-Java paths", conforming SQL language shall not contain an <SQL Java path>.

9.4 <routine invocation>

This Subclause modifies Subclause 10.4, "<routine invocation>", in ISO/IEC 9075-2.

Function

Invoke an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) $\boxed{\text{Insert after the last CR:}}$ Without Feature J611, "References", conforming SQL language shall not contain a <reference value expression>.

2) $\boxed{\text{Insert after the last CR:}}$ Without Feature J611, "References", conforming SQL language shall not contain a <right arrow>.

9.5 Java routine signature determination

Function

Specify rules for how a Java method's signature is determined if it is not explicitly specified, and how it is validated, based either on information specified when creating an external Java routine or external Java data type, or on contents of descriptors available when invoking an SQL routine.

Subclause Signature

```
"Java routine signature determination" [Syntax Rules] (
  Parameter: "ELEMENT",
  Parameter: "INDEX",
  Parameter: "SUBJECT"
)
```

ELEMENT — the syntactic element.

INDEX — the method specification index.

SUBJECT — the subject routine, if any.

Syntax Rules

1) Let *SE* be the *ELEMENT*, let *i* be the *INDEX*, and let *SR* be the *SUBJECT* in an application of the Syntax Rules of this Subclause.

2) Information needed by later rules of this Subclause is gathered based on the context in which this Subclause is executed, as follows.

   Case:

   a) If *SE* specifies <SQL-invoked routine>, then:

      i) Let *JN*, *JCLSN*, *JMN*, and *JPDL* respectively be the <jar name>, <Java class name>, <Java method name>, and <Java parameter declaration list> contained in <external Java reference string>.

      ii) Let *SPDL* be <SQL parameter declaration list>.

      iii) If <SQL-invoked routine> contains <schema procedure>, then

         Case:

         1) If DYNAMIC RESULT SETS *N* is specified for some *N* greater than 0 (zero), then let *DRSN* be *N*.

         2) Otherwise let *DRSN* be 0 (zero).

      iv) If <SQL-invoked routine> contains <schema function>, and <SQL-invoked routine> specifies a collection-returning external function, then:

         1) Let *RDPL* be a result data area parameter list that specifies a comma-separated list of <SQL parameter declaration>s that have <parameter mode> OUT; their <parameter type>s are defined to be those of the effective SQL parameter list entries *PN*+1 through *PN*+*FRN* as defined in Subclause 11.60, "<SQL-invoked routine>".

2) Append to *RDPL* a <comma> and an <SQL parameter declaration> whose <data type> is character string of implementation-defined (IV195) length and character set SQL_TEXT with <parameter mode> INOUT.

3) Append to *RDPL* a <comma> and an <SQL parameter declaration> whose <data type> is an exact numeric type with scale 0 (zero) and with <parameter mode> IN.

4) Append to *SPDL* a <comma> and *RPDL*, to create an <SQL parameter declaration list> containing the input parameters, the result data item parameter(s), and the save area and call type data items.

b) If *SE* specifies <user-defined type definition>, then:

i) Let *UDTD* be the <user-defined type definition>, let *UDTB* be the <user-defined type body> immediately contained in *UDTD*, and let *UDTN* be the <schema-resolved user-defined type name> immediately contained in *UDTB*.

ii) Let *JN* and *JCLSN* respectively be the <jar name> and <Java class name> contained in <external Java type clause> contained in *UDTB*.

iii) For the purposes of parameter mapping as defined in Subclause 4.6, "Parameter mapping", the remaining rules in this Subclause are performed as if the descriptor for the user-defined type defined by *UDTD* was already available in the SQL-session. That descriptor describes the type as having the name *UDTN*, being an external Java data type, and having the <jar and class name> specified in *UDTD*.

iv) Let MS_i be the *i*-th <method specification> in the <method specification list> contained by *UDTB*.

v) Let *SRT* be the SQL <data type> specified in the RETURNS clause of MS_i.

vi) Let *DRSN* be 0 (zero).

vii) If MS_i immediately contains <static field method spec>, then:

1) Let *QJFN* be the <qualified Java field name> of MS_i.

2) Let *FI* be the <Java identifier> contained in <Java field name> contained in *QJFN*.

3) If *QJFN* specifies a <Java class name>, then let *SFC* be that class name; otherwise, let *SFC* be *JCLSN*.

4) Let *SPDL* be the <SQL parameter declaration list>

```
( )
```

viii) If MS_i does not immediately contain <static field method spec>, then:

1) Let *JMN* and *JPDL* respectively be the <Java method name> and <Java parameter declaration list> contained in <Java method and parameter declarations> contained in MS_i.

2) Let *SPDL* be the augmented SQL parameter declaration list NPL_i of MS_i.

3) If MS_i specifies a collection-returning external function then:

A) Let *RDPL* be a result data area parameter list that specifies a comma-separated list of <SQL parameter declaration>s that have <parameter mode> OUT; their <parameter type>s are defined to be those of the effective SQL parameter list entries *PN*+1 through *PN*+*FRN* as defined in Subclause 10.8, "<SQL-invoked routine>".

B) Append to *RDPL* a <comma> and an <SQL parameter declaration> whose <data type> is character string of implementation-defined (IV195) length and character set SQL_TEXT with <parameter mode> INOUT.

C) Append to *RDPL* a <comma> and an <SQL parameter declaration> whose <data type> is an exact numeric type with scale 0 (zero) and with <parameter mode> IN.

D) Append to *SPDL* a <comma> and *RPDL*, to create an <SQL parameter declaration list> containing the augmented SQL declaration parameter list, the result data item parameter(s), and the save area and call type data items.

c) Otherwise, descriptors are available.

i) Let *SRD* be the routine descriptor of *SR*.

ii) If *SRD* indicates that the SQL-invoked routine is an SQL-invoked method, then:

1) Let *SRUDT* be the user-defined type whose descriptor contains *SR*'s corresponding method specification descriptor *MSD*, and let *SRUDTD* be the user-defined type descriptor of *SRUDT*.

2) Let *JN* and *JCLSN* respectively be the <jar name> and <Java class name> contained by *SRUDTD*'s <jar and class name>.

3) Let *SRT* be the SQL <returns data type> specified in *MSD*.

4) Let *DRSN* be 0 (zero).

5) If *MSD* indicates that it is a static field method, then:

A) Let *FI* be the <Java identifier> contained in the <Java field name> of *MSD*.

B) Let *SFC* be the <Java class name> of *MSD*.

C) Let *SPDL* be the <SQL parameter declaration list>

```
( )
```

6) If *MSD* indicates that it is not a static field method, then:

A) Let *JMN* and *JPDL* respectively be the Java method name composed of the package, class, and name of the Java routine contained in *MSD* and the Java parameter declaration list contained in the signature contained in *MSD*.

B) Let *SPDL* be the augmented SQL parameter declaration list of *MSD*.

iii) If *SRD* indicates that the SQL-invoked routine is an SQL-invoked procedure or an SQL-invoked regular function, then:

1) Let *JN*, *JCLSN*, *JMN*, and *JPDL* respectively be the <jar name>, <Java class name>, <Java method name>, and <Java parameter declaration list> contained in <external Java reference string> contained in the <external routine name> of *SRD*.

2) Let *SPDL* be an SQL parameter declaration list composed of the SQL-invoked routine's SQL parameters contained in *SRD*, specified with the descriptors list of the <SQL parameter name>, if specified, the <data type>, the ordinal position, and an indication of whether the SQL parameter is an input SQL parameter, an

output SQL parameter, or both an input SQL parameter and an output SQL parameter.

3) If the SQL-invoked routine is an SQL-invoked procedure, then let *DRSN* be the maximum number of dynamic result sets as indicated by *SRD*; otherwise, let *DRSN* be 0 (zero).

4) If the SQL-invoked routine is an SQL-invoked regular function that is not a collection-returning external function, then let *SRT* be the SQL <returns data type> specified in *MSD*; otherwise, let *SRT* be "`void`".

5) If the SQL-invoked routine is an SQL-invoked regular function that is a collection-returning external function, then:

   A) Let *RDPL* be a result data area parameter list that specifies a comma-separated list of <SQL parameter declaration>s that have <parameter mode> OUT; their <parameter type>s are defined to be those of the effective SQL parameter list entries *PN*+1 through *PN*+*FRN* as defined in Subclause 10.8, "<SQL-invoked routine>".

   B) Append to *RDPL* a <comma> and an <SQL parameter declaration> whose <data type> is character string of implementation-defined (IV195) length and character set SQL_TEXT with <parameter mode> INOUT.

   C) Append to *RDPL* a <comma> and an <SQL parameter declaration> whose <data type> is an exact numeric type with scale 0 (zero) and with <parameter mode> IN.

   D) Append to *SPDL* a <comma> and *RPDL*, to create an <SQL parameter declaration list> containing the input parameters, the result data item parameter(s), and the save area and call type data items.

3) Case:

a) If *JMN* is "`main`" and *SE* does not specify <user-defined type definition> or contain <method invocation>, then:

   i) If *SE* specifies <SQL-invoked routine>, then it shall contain <schema procedure> and shall not contain <returned result sets characteristic>.

   ii) If *SE* contains <routine invocation> then it shall contain <call statement>.

   iii) If a Java parameter declaration list *JPDL* is specified, then it shall be the following:

```
(java.lang.String[])
```

   iv) If a Java parameter declaration list is not specified, then let *JPDL* be the following:

```
(java.lang.String[])
```

   v) *SPDL* shall specify either:

      1) A single parameter that is an SQL ARRAY of CHARACTER or an ARRAY of CHARACTER VARYING. At runtime, this parameter is passed as a Java array of `java.lang.String`.

         NOTE 44 — This <SQL parameter declaration> can only be specified if the SQL system supports Feature S201, "SQL-invoked routines on arrays".

      2) Zero or more parameters, each of which is CHARACTER or CHARACTER VARYING. At runtime, these parameters are passed a Java array of `java.lang.String` (with possibly zero elements).

vi) Let *JCS* be the set of visible Java methods of class *JCLSN* in JAR *JN* whose method names are "`main`" and whose Java parameter data types list is *JPDL*.

NOTE 45 — "visible" is defined in Subclause 4.6, "Parameter mapping".

b) Otherwise:

i) Let *SPN* and *JPN* be, respectively, the number of <SQL parameter declaration>s in *SPDL* and the number of <Java data type>s in *JPDL*.

ii) If *JPDL* specifies a <Java parameter declaration list>, then:

1) If *i* is greater than 0 (zero) and MS_i specifies INSTANCE or CONSTRUCTOR or if *SRD* indicates the SQL-invoked routine is an SQL-invoked method and *MSD* indicates it is an instance method or a constructor, then prefix the Java parameter declaration list *JPDL* with the necessary subject parameter as follows.

Case:

A) If *JPDL* contains one or more <Java data type>s, then prefix the list of <Java data type>s immediately contained in <Java parameters> immediately contained in *JPDL* with

```
JCLSN,
```

B) Otherwise, replace *JPDL* with the <Java parameter declaration list>

```
 ( JCLSN )
```

2) For each <SQL parameter declaration> *SP* in *SPDL*, let *ST* be the <data type> of *SP* and let *JT* be the corresponding <Java data type> in *JPDL*.

A) If *SP* specifies IN, or does not specify an explicit <parameter mode>, then:

I) If *SP* is not an SQL array, then *JT* and *ST* shall be simply mappable or object mappable.

II) If *SP* is an SQL array, then *JT* and *ST* shall be array mappable.

B) If *SP* specifies OUT or INOUT, then

Case:

I) If *SPDL* has been augmented with a save area data item and *SP* is the *SPN*−1-th entry in the list (the save area data item), then *JT* and *ST* shall be output mappable or *JT* shall specify the class `java.lang.StringBuffer`.

II) Otherwise, *JT* and *ST* shall be output mappable.

NOTE 46 — "simply mappable", "object mappable", and "array mappable" are defined in Subclause 4.6, "Parameter mapping".

3) Case:

A) If *DRSN* is greater than 0 (zero), then *JPN* shall be greater than *SPN*, and each <Java data type> in *JPDL* whose ordinal position is greater than *SPN* shall be result set mappable.

B) Otherwise, *JPN* shall be equivalent to *SPN*.

iii) If a Java parameter declaration list is not specified, then determine the first *SPN* members of the Java parameter declaration list *JPDL* from *SPDL* as follows:

1) For each parameter *SP* of *SPDL* whose <parameter mode> is IN, or that does not specify an explicit <parameter mode>, if *SP* is not an SQL array, then let the corresponding Java parameter data type of *SP* be the corresponding Java data type of the <parameter type> of *SP*; if *SP* is an SQL array, then let *JT* be the corresponding Java data type of the <parameter type> of *SP*, and let the corresponding Java parameter data type of *SP* be an array of *JT*, that is, be `JT[]`.

   NOTE 47 — The "corresponding Java parameter data type" of *SP* is defined in Subclause 4.6, "Parameter mapping".

2) For each parameter *SP* of *SPDL* whose <parameter mode> is INOUT or OUT, let *JT* be the corresponding Java data type of the <parameter type> of *SP*, and let the corresponding Java parameter data type of *SP* be an array of *JT*, that is, be `JT[]`.

3) The <Java parameters> of *JPDL* is a list of the corresponding Java parameter data types of *SPDL*.

   NOTE 48 — *JPDL* does not specify parameter names. That is, the parameter names of the Java method do not have to match the SQL parameter names.

iv) The subject Java field of <static field method spec>s or the set of candidate visible Java methods are determined as follows.

Case:

1) If *SE* specifies <SQL-invoked routine> or if *SRD* indicates that the SQL-invoked routine is an SQL-invoked procedure or an SQL-invoked regular function, then:

   A) If *DRSN* is greater than 0 (zero), then:

      I) Let *SPN* and *JPN* be, respectively, the number of <SQL parameter declaration>s in *SPDL* and the number of <Java data type>s in *JPDL*.

      II) If *SPN* is equivalent to *JPN*, then *JPDL* was originally not specified; let *JCS* be the set of visible Java methods of class *JCLSN* in JAR *JN* whose method names are *JMN*, whose first *SPN* parameter data types are those of *JPDL*, and whose last *K* parameter data types, for some positive *K*, are result set mappable.

      III) If *SPN* is less than *JPN*, then *JPDL* was originally specified; let *JCS* be the set of visible Java methods of class *JCLSN* in JAR *JN* whose method names are *JMN*, whose Java parameter data types list is *JPDL*.

   B) If *DRSN* is 0 (zero), then let *JCS* be the set of visible Java methods of class *JCLSN* in JAR *JN* whose method names are *JMN*, whose Java parameter data types list is *JPDL*.

2) If *SE* specifies <user-defined type definition> or if *SRD* indicates that the SQL-invoked routine is an SQL-invoked method then:

   A) If *i* is greater than 0 (zero) and MS_i contains <static field method spec>, or if *MSD* indicates that it is a static field method, then:

      I) *FI* shall be the name of a field of *SFC*. Let *JSF* be that field.

      II) *JSF* shall be a public static field.

      III) Let *JFT* be the Java data type of *JSF*.

      IV) *SRT* and *JFT* shall be simply mappable or object mappable.

         NOTE 49 — "simply mappable" and "object mappable" are defined in Subclause 4.6, "Parameter mapping".

V) *JSF* is the subject static field of the SQL-invoked method defined by MS_i.

NOTE 50 — The subject Java class can contain fields and methods (public and private) for which no corresponding attribute or method is specified.

B) If *i* is greater than 0 (zero) and MS_i does not immediately contain <static field method spec>, or if *MSD* indicates that it is not a static field method, then:

I) Case:

1) If *i* is greater than 0 (zero) and MS_i specifies INSTANCE or CONSTRUCTOR, or if *MSD* indicates it is an instance method or a constructor, then *JPDL* contains the augmented Java parameter declaration list for this method. Remove the subject parameter from the Java parameter declaration list *JPDL* to create the unaugmented Java parameter declaration list *UAJPDL*, as follows.

Case:

a) If *JPDL* contains two or more <Java data type>s, then copy all *JPDL* to *UAJPDL*, omitting the first <Java data type> *JCLSN*, and its associated " , ".

b) Otherwise, set *UAJPDL* to the <Java parameter declaration list>

( )

2) Otherwise copy *JPDL* to *UAJPDL*.

II) Using Java overloading resolution, specified by *The Java Language Specification, Second Edition*, let *JCS* be the set of visible Java methods of class *JCLSN* in JAR *JN* or the supertypes of that class whose method names are *JMN* and whose Java parameter data types list is *UAJPDL*.

NOTE 51 — "visible" is defined in Subclause 4.6, "Parameter mapping".

III) If *i* is greater than 0 (zero) and MS_i specifies STATIC, or *MSD* indicates that STATIC was specified, then remove from *JCS* any Java method that is not static. Otherwise, remove from *JCS* any static Java method.

IV) If *i* is greater than 0 (zero) and MS_i specifies CONSTRUCTOR, or *MSD* indicates that CONSTRUCTOR was specified, then remove from *JCS* any Java method that is not a constructor. Otherwise, remove from *JCS* any Java method that is a constructor.

4) The subject Java method is determined as follows.

Case:

a) If *SE* specifies <SQL-invoked routine> or if *SRD* indicates that the SQL-invoked routine is an SQL-invoked procedure or an SQL-invoked regular function, then:

i) *JCS* shall contain exactly one Java method. Let *JM* be that Java method. The SQL-invoked routine is associated with *JM*.

ii) *JM* is the subject Java method of the SQL-invoked routine.

b) If *SE* specifies <user-defined type definition> or if *SRD* indicates that the SQL-invoked routine is an SQL-invoked method then, if *i* is greater than 0 (zero) and MS_i does not immediately contain <static field method spec>, or if *MSD* indicates that it is not a static field method then:

i) *JCS* shall contain exactly one Java method. Let *JM* be that Java method. The <Java method name> is referred to as the *corresponding Java method name* of <method name>.

ii) *JM* is the *subject Java method* of the SQL-invoked method.

5) The result data type of the SQL-invoked routine is validated as follows.

Case:

a) If *SE* specifies <SQL-invoked routine> or if *SRD* indicates that the SQL-invoked routine is an SQL-invoked procedure or an SQL-invoked regular function, then let *JRT* be the Java returns data type of *JM*.

i) If *JM* is an SQL-invoked procedure, then *JRT* shall be `void`.

ii) If *JM* is an SQL-invoked regular function that is not a collection-returning external function, then *JRT* and *SRT* shall be simply mappable or object mappable.

iii) If *JM* is a collection-returning external function, then *JRT* shall be `void`.

b) If *SE* specifies <user-defined type definition> or if *SRD* indicates that the SQL-invoked routine is an SQL-invoked method then, if *i* is greater than 0 (zero) and MS_i does not immediately contain <static field method spec>, or if *MSD* indicates that it is not a static field method, then let *JRT* be the Java returns data type of *JM*. If SELF AS RESULT is not specified then *JRT* and *SRT* shall be simply mappable or object mappable.

NOTE 52 — "simply mappable" and "object mappable" are defined in Subclause 4.6, "Parameter mapping".

c) Otherwise, let *JRT* be the Java data type of the subject static field. *JRT* and *SRT* shall be simply mappable or object mappable.

6) Evaluation of the Syntax Rules is terminated and control is returned to the invoking Subclause.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

10 Schema definition and manipulation

This Clause modifies Clause 11, "Schema definition and manipulation", in ISO/IEC 9075-2.

10.1 <drop schema statement>

This Subclause modifies Subclause 11.2, "<drop schema statement>", in ISO/IEC 9075-2.

Function

Destroy a schema.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 4)n): JARs.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1): If the SQL-implementation supports Feature J531, "Deployment", then:

   a) Let *JNS* be a <character string literal> containing the qualified <jar name> included in the descriptor of any JAR included in *S*.

   b) The following <call statement> is effectively executed:

```
CALL SQLJ.REMOVE_JAR ( JNS, 1 );
```

2) Insert after GR 12): If the SQL-implementation does not support Feature J531, "Deployment", then:

   a) Let *JNS* be a <character string literal> containing the qualified <jar name> included in the descriptor of any JAR included in *S*.

   b) The following <call statement> is effectively executed:

```
CALL SQLJ.REMOVE_JAR ( JNS, 0 );
```

Conformance Rules

No additional Conformance Rules.

10.2 <table definition>

This Subclause modifies Subclause 11.3, "<table definition>", in ISO/IEC 9075-2.

Function

Define a persistent base table, a created local temporary table, or a global temporary table.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 11)e): *ST* shall not be an external Java data type whose descriptor specifies an <interface specification> of SERIALIZABLE.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.3 <view definition>

This Subclause modifies Subclause 11.32, “<view definition>”, in ISO/IEC 9075-2.

Function

Define a viewed table.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 23)c): *ST* shall not be an external Java data type whose descriptor specifies an <interface specification> of SERIALIZABLE.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.4 <user-defined type definition>

This Subclause modifies Subclause 11.51, “<user-defined type definition>”, in ISO/IEC 9075-2.

Function

Define a user-defined type.

Format

```
<user-defined type body> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <schema-resolved user-defined type name> [ <subtype clause> ]
        <external Java type clause>
        [ AS <representation> ]
        [ <user-defined type option list> ] [ <method specification list> ]

<external Java type clause> ::=
  <external Java class clause> LANGUAGE JAVA <interface using clause>

<interface using clause> ::=
  [ USING <interface specification> ]

<interface specification> ::=
    SQLDATA
  | SERIALIZABLE

<method specification> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <static field method spec>

<method characteristic> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <external Java method clause>

<static field method spec> ::=
  STATIC METHOD <method name> <left paren> <right paren>
      <static method returns clause> [ SPECIFIC <specific method name> ]
      <external variable name clause>

<static method returns clause> ::=
  RETURNS <data type>

<external variable name clause> ::=
  EXTERNAL VARIABLE NAME <character string literal>

<external Java class clause> ::=
  EXTERNAL NAME <character string literal>

<external Java method clause> ::=
  EXTERNAL NAME <character string literal>

<Java method and parameter declarations> ::=
  <Java method name> [ <Java parameter declaration list> ]
```

Syntax Rules

1) Insert after SR 3): If <external Java type clause> is specified, then *UDT* is an *external Java data type*.

2) Insert after SR 8)j)ii): If *UDT* is an external Java data type, then *SST* shall be an external Java data type, and the subject Java class of *UDT* shall be a direct subclass of the subject Java class of *SST*. If *UDT* is not an external Java data type, then *SST* shall not be an external Java data type.

3) Insert before SR 9): If <external Java type clause> is specified, then:

   a) Let *VJC* be the value of the <character string literal> immediately contained in <external Java class clause>; *VJC* shall conform to the Format and Syntax Rules of<jar and class name>. The Java class identified by <Java class name> in the JAR identified by <jar id> in their immediately containing <jar and class name> is *UDT*'s *subject Java class*.

      NOTE 53 — The subject Java class of *UDT* can be the subject Java class of other external Java data types. Each such external Java data type is distinct from other such data types.

   b) *UDT*'s subject Java class shall be a `public` class and shall implement the Java interface `java.io.Serializable` or the Java interface `java.sql.SQLData` or both.

   c) If an <interface using clause> is not explicitly specified, then an implementation-defined (ID230) <interface specification> is implicit.

   d) If SERIALIZABLE is specified, then the subject Java class shall implement the Java interface `java.io.Serializable`. The method `java.io.Serializable.writeObject()` is effectively used to convert a Java object to an SQL representation, and the method `java.io.Serializable.readObject()` is effectively used to convert an SQL representation to a Java object.

   e) If SQLDATA is specified, then the subject Java class shall implement the Java interface `java.sql.SQLData` as defined in JDBC and JVMS. The method `java.sql.SQLData.writeSQL()` is effectively used to convert a Java object to an SQL representation, and the method`java.sql.SQLData.readSQL()` is effectively used to convert an SQL representation to a Java object.

   f) <overriding method specification> shall not be specified.

   g) A <representation> that is a <predefined type> shall not be specified.

   h) SELF AS LOCATOR shall not be specified.

   i) <locator indication> shall not be specified.

4) Insert before SR 9): If <external Java type clause> is not specified, then:

   a) <method specification> shall not specify <static field method spec>.

   b) <method characteristic> shall not specify <external Java method clause>.

   c) The <language clause> immediately contained in <method characteristic> shall not specify JAVA.

5) Insert after SR 9)a): If *UDT* is an external Java data type, then it is implementation-defined (IA144) whether validation of the explicit or implicit <Java parameter declaration list> is performed by <user-defined type definition> or when the corresponding SQL-invoked method is invoked.

6) Insert after SR 9)b)iii)6): If *UDT* is an external Java data type, then the <Java identifier> immediately contained in <Java method name> of MS_i shall be equivalent to the <Java identifier> immediately contained in the <class identifier> immediately contained in <jar and class name> of *UDT*.

7) Insert after SR 9)b)ix)4)B): *UDT* shall not be an external Java data type.

8) Insert after SR 9)b)x)3): *UDT* shall not be an external Java data type.

9) Insert after SR 9)b)xiii): If MS_i specifies <static field method spec>, then:

a) MS_i specifies a *static field method*.

b) Let *VSF* be the value of the <character string literal> simply contained in <static field method spec>; *VSF* shall conform to the Format and Syntax Rules of <qualified Java field name>.

NOTE 54 — <static field method spec> defines a static method of the user-defined type that returns the value of the Java static field specified by the <qualified Java field name>. This is a shorthand that provides read-only SQL access to static fields of the subject Java class or a superclass of the subject Java class.

10) Insert after SR 9)b)xiv)1): If *UDT* is an external Java data type then, with the exception of the implicit <original method specification>s generated for the observer and mutator functions of each attribute, the <method characteristics> of MS_i shall not contain the <method characteristic>s <language clause> or <parameter style clause> and shall contain exactly one <external Java method clause>. For an external Java data type, both <language clause> and <parameter style clause> implicitly specify JAVA.

11) Convert SR 9)b)xiv)2) to be: Case:

a) If *UDT* is an external Java data type, then let *VMP* be the value of the <character string literal> immediately contained in <external Java method clause>; *VMP* shall conform to the Format and Syntax Rules of <Java method and parameter declarations>.

b) Otherwise, the original SR 9)b)xiv)2)

12) Augment SR 9)b)xiv)6)B)I) by adding "UDT is not an external Java data type " as an additional restriction to the predicate of the rule.

13) Insert after SR 9)b)xv): If *UDT* is an external Java data type and validation of the <Java parameter declaration list> has been implementation-defined (IA144) to be performed by <user-defined type definition>, then the Syntax Rules of Subclause 9.5, "Java routine signature determination", are applied with <user-defined type definition> as *ELEMENT*, *i* as *INDEX*, and no subject routine as *SUBJECT*.

Access Rules

No additional Access Rules.

General Rules

1) Augment GR 1)g)xi) by adding "JAVA " to the set of languages that <language name> is not.

Conformance Rules

1) Insert after the last CR: Without Feature J511, "Commands", conforming SQL language shall not contain a <user-defined type definition> that contains an <external Java type clause> that is not contained in a <descriptor file>.

2) Insert after the last CR: Without Feature J591, "Overloading", conforming SQL language shall not contain a <method specification> that contains a <method name> that is equivalent to the <method name> of any other <method specification> in the same <user-defined type definition>.

3) Insert after the last CR: Without Feature J641, "Static fields", conforming SQL language shall not contain a <static field method spec>.

4) Insert after the last CR: Without Feature J541, "SERIALIZABLE" conforming SQL language shall not contain an <interface specification> that contains SERIALIZABLE.

5) Insert after the last CR: Without Feature J551, “SQLDATA”, conforming SQL language shall not contain an <interface specification> that contains SQLDATA.

6) Insert after the last CR: Without Feature J622, “external Java types”, conforming SQL language shall not contain a <user-defined type definition> that contains an <external Java type clause>.

10.5 <attribute definition>

This Subclause modifies Subclause 11.52, “<attribute definition>”, in ISO/IEC 9075-2.

Function

Define an attribute of a structured type.

Format

```
<attribute definition> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <attribute name> <data type>
        [ <attribute default> ]
        [ <collate clause> ] <external Java attribute clause>

<external Java attribute clause> ::=
  EXTERNAL NAME <character string literal>
```

Syntax Rules

1) Insert after SR 1): If the <attribute definition> is contained in a <user-defined type definition> that is not an external Java data type or is contained in an <alter type statement>, then <attribute definition> shall not specify an <external Java attribute clause>.

2) Insert after SR 1): If the <attribute definition> is contained in a <user-defined type definition> that specifies an external Java data type whose <interface specification> is SERIALIZABLE, then <attribute definition> shall specify an <external Java attribute clause>.

3) Insert after SR 1): If an <external Java attribute clause> is specified, then let *VFN* be the value of the <character string literal> immediately contained in <attribute definition>; *VFN* shall conform to the Format and Syntax Rules of <Java field name>. The <Java field name> value of *VFN* is referred to as the *corresponding Java field name* of the <attribute name>.

4) Insert after SR 1): If <attribute definition> is contained in a <user-defined type definition> that specifies an external Java data type, then <attribute default> and <collate clause> shall not be specified.

5) Insert after SR 1): If <attribute definition> is contained in a <user-defined type definition> that specifies an external Java data type, and if the <data type> specified in the <attribute definition> is a structured type *ST*, then *ST* shall be an external Java data type.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 4)e): If the <attribute definition> contains an <external Java attribute clause>, then the corresponding Java field name of the <attribute name>.

2) Insert before GR 5)b):

   a) If *UDT* is not an external Java data type whose descriptor’s <interface specification> specifies SERIALIZABLE, then `V.AN()` returns the value of *A* in *V*.

b) If *UDT* is an external Java data type whose descriptor's <interface specification> specifies SERIALIZABLE, then the `readObject()` method of the subject Java class *SJCE* of *V* is effectively used to obtain a Java object from the value of *V*, the Java field that corresponds to the attribute specified in <Java field name> contained by <attribute definition> is accessed. Let *JV* and *JCLS* be respectively that Java value and its most specific Java class.

Case:

i) If *DT* is a user-defined type, then:

1) Let *STU* be the user-defined type whose subject Java class is *JCLS* and whose user-defined type is *DT* or is a subclass of *DT*.

2) Let *UIS* be the <interface specification> specified by the user-defined type descriptor of *STU*.

3) Case:

A) If *UIS* is SERIALIZABLE, then:

I) The subject Java class *JCLS*'s `writeObject()` method is executed to convert the Java value *JV* to the SQL value *SV* of user-defined type *STU*.

II) The method of execution of the subject Java class's implementation of `writeObject()` is implementation-defined (IW170).

NOTE 55 — If *UIS* is SERIALIZABLE, then, as described in Subclause 10.4, "<user-defined type definition>", the descriptor's subject Java class implements the Java interface `java.io.Serializable` and defines that interface's `writeObject()` method as described by JVMS.

B) If *UIS* is SQLDATA, then:

I) The subject Java class *JCLS*'s `writeSQL()` method is executed to convert the Java value *JV* to the SQL value *SV* of user-defined type *STU*.

II) The method of execution of the subject Java class's implementation of `writeSQL()` is implementation-defined (IW171).

NOTE 56 — If *UIS* is SQLDATA, then, as described in Subclause 10.4, "<user-defined type definition>", the descriptor's subject Java class implements the Java interface `java.sql.SQLData` and defines that interface's `writeSQL()` method as described by JDBC and JVMS.

C) Otherwise, the value of *SV* is set to the value of *JV*.

3) Insert before GR 6)b):

a) If *UDT* is an external Java data type whose descriptor's <interface specification> specifies SERIALIZABLE, then the `readObject()` method of the subject Java class *SJCE* of *V* is effectively used to obtain a Java object from the value of *V*. Let *MST*, *JCLS*, and *Jtemp* be respectively the most specific type of *AV*, the subject Java class of *MST*, and the Java object obtained from `readObject()`.

i) Case:

1) If *MST* is a user-defined type, then:

A) Let *UIS* be the <interface specification> specified by the user-defined type descriptor of *MST*.

B) Case:

I) If *UIS* is SERIALIZABLE, then:

1) The subject Java class *JCLS*'s `readObject()` method is executed to convert the value of *AV* to a Java object *JV*.

2) The method of execution of the subject Java class's implementation of `readObject()` is implementation-defined (IW172).

NOTE 57 — If *UIS* is SERIALIZABLE, then, as described in Subclause 10.4, "<user-defined type definition>", the subject Java class of *U* implements the Java interface `java.io.Serializable` and defines that interface's `readObject()` method as described by JVMS.

II) If *UIS* is SQLDATA, then:

1) The subject Java class *JCLS*'s `readSQL()` method is executed to convert the value of *AV* to a Java object *JV*.

2) The method of execution of the subject Java class's implementation of `readSQL()` is implementation-defined (IW173).

NOTE 58 — If *UIS* is SQLDATA, then, as described in Subclause 10.4, "<user-defined type definition>", the subject Java class of *U* implements the Java interface `java.sql.SQLData` and defines that interface's `readSQL()` method as described by JDBC and JVMS.

2) Otherwise, the value of *JV* is set to the value of *AV*.

ii) The Java field of *Jtemp* that corresponds to the attribute specified in <Java field name> contained by <attribute definition> is assigned the value *JV*.

iii) The subject Java class *SJCE* of *V*'s `writeObject()` method is effectively used to obtain an SQL value *V2* from the Java value *Jtemp*.

iv) The invocation `V.AN(AV)` returns *V2*.

Conformance Rules

No additional Conformance Rules.

10.6 <alter type statement>

This Subclause modifies Subclause 11.53, “<alter type statement>”, in ISO/IEC 9075-2.

Function

Change the definition of a user-defined type.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 1): *D* shall not be an external Java data type.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.7 <drop data type statement>

This Subclause modifies Subclause 11.59, "<drop data type statement>", in ISO/IEC 9075-2.

Function

Destroy a user-defined type.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert after the last CR: Without Feature J511, "Commands", conforming SQL language shall not contain a <drop data type statement> that contains a <schema-resolved user-defined type name> that identifies an external Java type and that is not contained in a <descriptor file>.

2) Insert after the last CR: Without Feature J622, "external Java types", conforming SQL language shall not contain a <drop data type statement> that contains a <schema-resolved user-defined type name> that identifies an external Java type.

10.8 <SQL-invoked routine>

This Subclause modifies Subclause 11.60, "<SQL-invoked routine>", in ISO/IEC 9075-2.

Function

Define an SQL-invoked routine.

Format

```
<parameter style> ::=
    !! All alternatives from ISO/IEC 9075-2
  | JAVA

<external Java reference string> ::=
  <jar and class name> <period> <Java method name>
      [ <Java parameter declaration list> ]
```

Syntax Rules

1) Insert after SR 3): If <SQL-invoked routine> specifies LANGUAGE JAVA, then no <SQL parameter declaration> specified in <SQL-invoked function> shall specify RESULT.

2) Insert after SR 3): If <SQL-invoked routine> specifies LANGUAGE JAVA, then neither the <returns clause> contained in <SQL-invoked function> nor any <SQL parameter declaration> contained in an <SQL-invoked function> or <SQL-invoked procedure> shall contain <locator indication>.

3) Insert after SR 3): If <SQL-invoked routine> specifies LANGUAGE JAVA, then <transform group specification> shall not be specified.

4) Insert after SR 3): The maximum value of <maximum returned result sets> is implementation-defined (IL204).

5) Insert after SR 8)b)ii): *UDT* shall not be an external Java type.

6) Insert after SR 9)a): If LANGUAGE JAVA is specified, then <parameter style clause> shall specify <parameter style> JAVA.

7) Insert after SR 9)i): An external routine that specifies LANGUAGE JAVA is called an *external Java routine*.

8) Insert after SR 9)i): If *R* is an external Java routine, then the <external routine name> immediately contained in <external body reference> shall specify a <character string literal>. Let *V* be the value of that <character string literal>. *V* shall conform to the Format and Syntax Rules of0 an <external Java reference string>.

   NOTE 59 — *R* is defined by ISO/IEC 9075-2 to be the SQL-invoked routine specified by <SQL-invoked routine>.

9) Convert SR 9)x)ii) to be: Case:

   a) If *R* is an external Java routine, then the <Java method name> is the name of one or more Java methods in the class specified by <Java class name> in the JAR specified by <jar name>. The combination of <Java class name> and <Java method name> represent a fully qualified Java class name and method name. The method name can reference a method of the class, or a method of a superclass of the class.

   b) Otherwise, the original SR 9)x)ii)

10) Augment SR 8) by adding "If *R* is not an external Java routine " as a restriction to the text of the Rule.

11) Insert before SR 24)e): If PARAMETER STYLE JAVA is specified, then:

a) Case:

i) If *R* is a collection-returning external function and the returned array's element type or returned multiset's element type is a row type, then let *FRN* be the degree of the element type.

ii) Otherwise, let *FRN* be 1 (one).

b) If *R* is a collection-returning external function, then let *AREF* be 2. Otherwise, let *AREF* be 0 (zero).

c) If *R* is an SQL-invoked function, then let the *effective SQL parameter list* be a list of *PN*+*FRN*+*AREF* SQL parameters, as follows:

i) For *i* ranging from 1 (one) to *PN*, the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration>.

ii) Case:

1) If *FRN* is 1 (one), then effective SQL parameter list entry *PN*+*FRN* has <parameter mode> OUT; its <parameter type> *PT* is defined as follows:

A) If <result cast> is specified, then let *RT* be <result cast from type>; otherwise, let *RT* be <returns data type>.

B) If *R* is a collection-returning external function, then let *PT* be the element type of *RT*.

C) If *R* is not a collection-returning external function, then *PT* is *RT*.

2) Otherwise, for *i* ranging from *PN*+1 to *PN*+*FRN*, the *i*-th effective SQL parameter list entry is defined as follows:

A) Its <parameter mode> is OUT.

B) Let RFT_{i-PN} be the data type of the (*i*–*PN*)-th field of the element type of the <returns data type>. The <parameter type> PT_i of the *i*-th effective SQL parameter list entry is RFT_{i-PN}.

iii) If *R* is a collection-returning external function, then:

1) Effective SQL parameter type list entry (*PN*+*FRN*)+1 is an SQL parameter whose <data type> is character string of implementation-defined (IV195) length and character set SQL_TEXT with <parameter mode> INOUT.

2) Effective SQL parameter type list entry (*PN*+*FRN*)+2 is an SQL parameter whose <data type> is an exact numeric type with scale 0 (zero) and with <parameter mode> IN.

d) If *R* is an SQL-invoked procedure, then let the effective SQL parameter list be a list of *PN* SQL parameters. For *i* ranging from 1 (one) to *PN*, the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration>.

12) Augment SR 24)g) by adding "If <language clause> does not specify JAVA " as a restriction to the Rule.

13) Augment SR 24)h) by adding "<language clause> does not specify JAVA " as an additional restriction to the predicate of the Rule.

14) Insert before SR 25):

NOTE 60 — The rules for parameter type correspondence when LANGUAGE JAVA is specified are given in Subclause 4.6, "Parameter mapping".

15) Insert before SR 25): If *R* is an external Java routine, then it is implementation-defined (IA144) whether validation of the explicit or implicit <Java parameter declaration list> is performed by <SQL-invoked routine> or when its SQL-invoked routine is invoked.

16) Insert before SR 25): If *R* is an external Java routine, and validation of the <Java parameter declaration list> has been implementation-defined (IA144) to be performed by <SQL-invoked routine>, then the Syntax Rules of Subclause 9.5, "Java routine signature determination", are applied with <SQL-invoked routine> as *ELEMENT*, 0 (zero) as *INDEX*, and no subject routine as *SUBJECT*.

Access Rules

1) Insert after AR 1): If *R* is an external Java routine, then the applicable privileges for *A* shall include USAGE privilege on the JAR referenced in the <external Java reference string>.

NOTE 61 — The references to *R* and *A* are defined in the Syntax Rules of Subclause 11.60, "<SQL-invoked routine>", in ISO/IEC 9075-2.

General Rules

1) Insert into GR 3)m)ii), after the last list item:

   a) PARAMETER STYLE JAVA

2) Augment GR 6)a)i) by adding "*R* is not an external Java routine " as an additional restriction to the predicate in the lead text of the Rule.

Conformance Rules

1) Insert after the last CR: Without Feature J511, "Commands", conforming SQL language shall not contain an <SQL-invoked routine> that contains a <language name> that contains JAVA and that is not contained in a <descriptor file>.

2) Insert after the last CR: Without Feature J581, "Output parameters", conforming SQL language shall not contain an <SQL-invoked routine> that contains a <language name> that contains JAVA and that contains a <parameter mode> that contains either OUT or INOUT.

3) Insert after the last CR: Without Feature J521, "JDBC data types", conforming SQL language shall not contain a <Java data type> that is not the corresponding Java data type of some SQL data type.

4) Insert after the last CR: Without Feature J621, "external Java routines", conforming SQL language shall not contain an <SQL-invoked routine> that contains a <language name> that contains JAVA.

10.9 <alter routine statement>

This Subclause modifies Subclause 11.61, "<alter routine statement>", in ISO/IEC 9075-2.

Function

Alter a characteristic of an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 1): *SR* shall not be an external Java routine.

NOTE 62 — *SR* is defined to be the SQL-invoked routine identified by the <alter routine statement>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10.10 <drop routine statement>

This Subclause modifies Subclause 11.62, "<drop routine statement>", in ISO/IEC 9075-2.

Function

Destroy an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

1) Insert after the last SR: If *SR* is an external Java routine and <drop routine statement> is contained in a <descriptor file>, then <drop routine statement> shall specify a <routine type> of PROCEDURE or of FUNCTION.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert after the last CR: Without Feature J511, "Commands", conforming SQL language shall not contain a <drop routine statement> that contains a <specific routine designator> that identifies an external Java routine and that is not contained in a <descriptor file>.

2) Insert after the last CR: Without Feature J621, "external Java routines", conforming SQL language shall not contain a <drop routine statement> that contains a <specific routine designator> that identifies an external Java routine.

10.11 <user-defined ordering definition>

This Subclause modifies Subclause 11.65, “<user-defined ordering definition>”, in ISO/IEC 9075-2.

Function

Define a user-defined ordering for a user-defined type.

Format

```
<ordering category> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <comparable category>

<comparable category> ::=
  RELATIVE WITH COMPARABLE INTERFACE
```

Syntax Rules

1) Insert into SR 4), after the last list item:
   a) <comparable category>
2) Convert SR 6)b) to be: Case:
   a) If <comparable category> is specified, then *UDT* shall be an external Java data type. Let *JC* be the subject Java class of that external Java data type. *JC* shall implement the Java interface `java.lang.Comparable`.
   b) Otherwise, the original SR 6)b).

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 3)c): If <comparable category> is specified, then the ordering category in the user-defined type descriptor of *UDT* is set to COMPARABLE.

Conformance Rules

1) Insert after the last CR: Without Feature J622, “external Java types”, conforming SQL language shall not contain a <user-defined ordering definition> that contains a <schema-resolved user-defined type name> that identifies an external Java type.
2) Insert after the last CR: Without Feature J511, “Commands”, conforming SQL language shall not contain a <user-defined ordering definition> that contains a <schema-resolved user-defined type name> that identifies an external Java type and that is not contained in a <descriptor file>.

10.12 <drop user-defined ordering statement>

This Subclause modifies Subclause 11.66, “<drop user-defined ordering statement>”, in ISO/IEC 9075-2.

Function

Destroy a user-defined ordering method.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert after the last CR: Without Feature J622, “external Java types”, conforming SQL language shall not contain a <drop user-defined ordering statement> that contains a <schema-resolved user-defined type name> that identifies an external Java type.

2) Insert after the last CR: Without Feature J511, “Commands”, conforming SQL language shall not contain a <drop user-defined ordering statement> that contains a <schema-resolved user-defined type name> that identifies an external Java type and that is not contained in a <descriptor file>.

11 Access control

This Clause modifies Clause 12, "Access control", in ISO/IEC 9075-2.

11.1 <grant privilege statement>

This Subclause modifies Subclause 12.2, "<grant privilege statement>", in ISO/IEC 9075-2.

Function

Define privileges.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert after the last CR: Without Feature J511, "Commands", conforming SQL language shall not contain a <grant privilege statement> that contains an <object name> that immediately contains a <jar name> and that is not contained in a <descriptor file>.

11.2 <privileges>

This Subclause modifies Subclause 12.3, "<privileges>", in ISO/IEC 9075-2.

Function

Specify privileges.

Format

```
<object name> ::=
    !! All alternatives from ISO/IEC 9075-2
  | JAR <jar name>
```

Syntax Rules

1) Insert after SR 4)a): If *ON* specifies a <jar name>, then *AC* shall specify USAGE.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert after the last CR: Without Feature J561, "JAR privileges", conforming SQL language shall not contain an <object name> that immediately contains a <jar name>.

11.3 <revoke statement>

This Subclause modifies Subclause 12.7, “<revoke statement>”, in ISO/IEC 9075-2.

Function

Destroy privileges and role authorizations.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Augment GR 2)b)i)3)D) by adding “JAR” to both lists of objects.

2) Insert after GR 12)j)ii): *DT* is an external Java data type and the revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE on the JAR whose <jar name> is contained in the <jar and class name> of the descriptor of *DT*.

3) Insert after GR 12)n): Let *JR* be any JAR descriptor included in *S1*. *JR* is said to be *impacted* if the revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on a JAR whose name is contained in a <resolution jar> contained in the SQL-Java path of *JR*.

4) Insert after GR 12)o)i)19): If *RD* is an external Java routine, USAGE on the JAR whose <jar name> is contained in <external Java reference string> contained in the <external routine name> of the descriptor of *RD*.

5) Insert after GR 12)q): If RESTRICT is specified, and there exists an impacted JAR, then an exception condition is raised: *dependent privilege descriptors still exist (2B000)*.

6) Insert after GR 12)ah): If the object identified by <object name> of the <revoke statement> specifies <jar name>, let *J* be the JAR identified by that <jar name>. For every impacted JAR descriptor *JR* and for each <path element> *PE* contained in the SQL-Java path of *JR* whose immediately contained <resolution jar> is *J*, the SQL-Java path of the JAR descriptor *JR* is modified such that it does not contain *PE*.

Conformance Rules

1) Insert after the last CR: Without Feature J511, “Commands”, conforming SQL language shall not contain a <revoke statement> that an <object name> that immediately contains a <jar name> and that is not contained in a <descriptor file>.

12 Built-in procedures

12.1 SQLJ.INSTALL_JAR procedure

Function

Install a set of Java classes into the current SQL catalog and schema.

Signature

```
SQLJ.INSTALL_JAR (
    url       IN    CHARACTER VARYING(L),
    jar       IN    CHARACTER VARYING(L),
    deploy    IN    INTEGER )
```

Where *L* is an implementation-defined (IL015) integer value.

Syntax Rules

None.

Access Rules

1) The privileges required to invoke the `SQLJ.INSTALL_JAR` procedure are implementation-defined (IW203).

General Rules

1) The `SQLJ.INSTALL_JAR` procedure is subject to implementation-defined (IA145) rules for executing SQL-schema statements within SQL-transactions. If an invocation of `SQLJ.INSTALL_JAR` raises an exception condition, then the effect on the install actions is implementation-defined (IA146).

2) The values of the `url` parameter that are valid are implementation-defined (IA147), and may include URLs whose format is implementation-defined (IA147). If the value of the `url` parameter does not conform to implementation-defined (IA147) restrictions and does not identify a valid JAR, then an exception condition is raised: *Java-related condition — invalid URL (46001)*.

3) Let *J* be the value of the `jar` parameter. Let *TJ* be the value of

   ```
   TRIM ( BOTH ' ' FROM J )
   ```

   If *TJ* does not conform to the Format and Syntax Rules of <jar name>, then an exception condition is raised: *Java-related condition — invalid JAR name (46002)*.

4) Let *JN* be the explicitly or implicitly qualified <jar id> specified in the <jar name> specified by *TJ*.

5) If there is an installed JAR whose name is *JN*, then an exception condition is raised: *Java-related condition — invalid JAR name (46002)*.

6) The JAR is installed and associated with the name *JN*. All contents of the JAR are installed, including both visible and non-visible Java classes, and other items contained in the JAR. This JAR becomes the *associated JAR* of each new class. The non-visible Java classes and other items can be referenced by other Java methods.

7) A JAR descriptor is created that describes the JAR being installed. The JAR descriptor includes the name of the JAR, and an empty SQL-Java path.

8) A privilege descriptor is created that defines the USAGE privilege on the JAR identified by the `jar` parameter to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the `jar` parameter. The grantor for the privilege descriptor is set to the special grantor value "_SYSTEM". The privilege is grantable.

9) If the value of the `deploy` parameter is not zero, and if the JAR contains one or more deployment descriptor files, then the install actions implied by those instances are performed in the order in which the deployment descriptor files appear in the manifest.

   NOTE 63 — Deployment descriptor files and their install actions are specified in Subclause 4.12.2, “Deployment descriptor files”.

Conformance Rules

1) Without Feature J531, “Deployment”, conforming SQL language shall not contain invocations of the SQLJ.INSTALL_JAR procedure that provide non-zero values of the `deploy` parameter.

12.2 SQLJ.REPLACE_JAR procedure

Function

Replace an installed JAR.

Signature

```
SQLJ.REPLACE_JAR (
    url      IN    CHARACTER VARYING (L),
    jar      IN    CHARACTER VARYING (L) )
```

Where *L* is an implementation-defined (IL015) integer value.

Syntax Rules

None.

Access Rules

1) The privileges required to invoke the `SQLJ.REPLACE_JAR` procedure are implementation-defined (IW204).

2) The current user shall be the owner of the JAR specified by the value of the `jar` parameter.

General Rules

1) The `SQLJ.REPLACE_JAR` procedure is subject to implementation-defined (IA148) rules for executing SQL-schema statements within SQL-transactions.

2) The values of the `url` parameter that are valid are implementation-defined (IA149), and may include URLs whose format is implementation-defined (IA149). If the value of `url` identifies a valid JAR, then refer to the classes in that JAR as the *new classes*. If the value of the `url` parameter does not identify a valid JAR, then an exception condition is raised: *Java-related condition — invalid URL (46001)*.

3) Let *J* be the value of the `jar` parameter. Let *TJ* be the value of

```
TRIM ( BOTH ' ' FROM J )
```

If *TJ* does not conform to the format of <jar name>, then an exception condition is raised: *Java-related condition — invalid JAR name (46002)*.

4) Let *JN* be the explicitly or implicitly qualified <jar id> specified in the <jar name> specified by *TJ*.

5) If there is an installed JAR with <jar name> *JN*, then refer to that JAR as the *old JAR*. Refer to the classes in the old JAR as the *old classes*. If there is not an installed JAR with <jar name> *JN*, then an exception condition is raised: *Java-related condition — attempt to replace uninstalled JAR (4600A)*. Equivalence of JAR names is determined by the rules for equivalence of identifiers as specified in Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

6) Let the *matching old classes* be the old classes whose fully qualified class names are the names of new classes and let the *matching new classes* be the new classes whose fully qualified class names

are the names of old classes. Let the *unmatched old classes* be the old classes that are not matching old classes and let the *unmatched new classes* be the new classes that are not matching new classes.

7) Let the *dependent SQL routines* of a JAR be the routines whose descriptor's <external routine name> specifies an <external Java reference string> whose immediately contained <jar name> is equivalent to the JAR name of that JAR.

8) If any dependent SQL routine of the old JAR references a method in an unmatched old class, then an exception condition is raised: *Java-related condition — invalid class deletion (46003)*.

   NOTE 64 — This rule prohibits deleting classes that are referenced by external Java routines. This prohibition does not, however, prevent deletion of classes that are referenced only indirectly by other Java classes.

9) For each dependent SQL routine of the old JAR that references a method in a matching old class, let *CS* be the <SQL-invoked routine> that created the SQL routine. If *CS* is not a valid <SQL-invoked routine> for the corresponding new routine, then an exception condition is raised: *Java-related condition — invalid replacement (46005)*.

10) Let the *dependent SQL types* of a JAR file be the external Java data types that have as their subject Java class a Java class contained in that JAR.

   NOTE 65 — "subject Java class" is defined in Subclause 10.4, "<user-defined type definition>".

11) If there are any dependent SQL types of the specified JAR file that are unmatched old classes, then an exception condition is raised: *Java-related condition — invalid class deletion (46003)*.

   NOTE 66 — This rule prohibits deleting classes that are referenced by external Java data types. This prohibition does not, however, prevent deletion of classes that are referenced only indirectly by other Java classes.

12) For each dependent SQL type, let *CT* be the <user-defined type definition> that created the SQL type. If *CT* is not a valid <user-defined type definition> for the corresponding new class, then an exception condition is raised: *Java-related condition — invalid replacement (46005)*.

13) The old JAR and all visible and non-visible old classes contained in it are deleted.

14) The new JAR and all visible and non-visible new classes are installed and associated with the specified <jar name>. That JAR becomes the *associated JAR* of each new class. All contents of the new JAR are installed, including both visible and non-visible Java classes, and other items contained in the JAR. The non-visible Java classes and other items can be referenced by other Java methods.

15) The effect of `SQLJ.REPLACE_JAR` on currently executing SQL statements that use an SQL routine or structured type whose implementation has been replaced is implementation-dependent (UA007).

Conformance Rules

None.

12.3 SQLJ.REMOVE_JAR procedure

Function

Remove an installed JAR and its classes.

Signature

```
SQLJ.REMOVE_JAR (
    jar      IN    CHARACTER VARYING (L),
    undeploy IN    INTEGER )
```

Where *L* is an implementation-defined (IL015) integer value.

Syntax Rules

None.

Access Rules

1) The privileges required to invoke the `SQLJ.REMOVE_JAR` procedure are implementation-defined (IW205).

2) The current user shall be the owner of the JAR specified by the value of the `jar` parameter.

General Rules

1) The `SQLJ.REMOVE_JAR` procedure is subject to implementation-defined (IA150) rules for executing SQL-schema statements within SQL-transactions. If an invocation of `SQLJ.REMOVE_JAR` raises an exception condition, then the effect on the remove actions is implementation-defined (IA151).

2) Let *J* be the value of the `jar` parameter. Let *TJ* be the value of

```
TRIM ( BOTH ' ' FROM J )
```

If *TJ* does not conform to the format of <jar name>, then an exception condition is raised: *Java-related condition — invalid JAR name (46002)*.

3) Let *JN* be the explicitly or implicitly qualified <jar id> specified in the <jar name> specified by *TJ*.

4) If there is not an installed JAR with <jar name> *JN*, then an exception condition is raised: *Java-related condition — attempt to remove uninstalled JAR (4600B)*. Equivalence of <jar name>s is determined by the rules for equivalence of identifiers as specified in Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

5) Let *JR* be the JAR identified by *JN*.

6) If the value of the `undeploy` parameter is not 0 (zero), and if *JR* contains one or more deployment descriptor files, then the remove actions implied by those instances are performed in the reverse of the order in which the deployment descriptor files appear in the manifest.

NOTE 67 — Deployment descriptor files and their remove actions are specified in Subclause 4.12.2, "Deployment descriptor files".

NOTE 68 — These actions are performed prior to examining the condition specified in the following step.

7) Let the *dependent SQL routines* of a JAR be the routines whose descriptor's <external routine name> specifies an <external Java reference string> whose immediately contained <jar name> is equivalent to the name of that JAR.

8) If there are any dependent SQL routines of *JR*, then an exception condition is raised: *Java-related condition — invalid class deletion (46003)*.

NOTE 69 — This rule prohibits deleting classes that are referenced by external Java routines. This prohibition does not, however, prevent deletion of classes that are referenced only indirectly by other Java classes.

9) Let the *dependent SQL types* of a JAR be the external Java data types that have as their subject Java class a Java class contained in that JAR.

NOTE 70 — "Subject Java class" is defined in Subclause 10.4, "<user-defined type definition>".

10) If there are any dependent SQL types of *JR*, then an exception condition is raised: *Java-related condition — invalid class deletion (46003)*.

NOTE 71 — This rule prohibits deleting classes that are referenced by external Java data types. This prohibition does not, however, prevent deletion of classes that are referenced only indirectly by other Java classes.

11) Let the *dependent JARs* of a JAR be the JARs whose descriptors specify an SQL-Java path that immediately contains one or more <path element>s whose contained <jar name> is equivalent to the name of that JAR.

12) If there are any dependent JARs of *JR*, then an exception condition is raised: *Java-related condition — invalid JAR removal (4600C)*.

13) *JR* and all visible and non-visible classes contained in it are deleted.

14) The USAGE privilege on *JR* is revoked from all users that have it.

15) The descriptor of *JR* is destroyed.

16) The effect of `SQLJ.REMOVE_JAR` on currently executing SQL statements that use an SQL routine or structured type whose implementation has been removed is implementation-dependent (UA008).

Conformance Rules

1) Without Feature J531, "Deployment", conforming SQL language shall not contain invocations of the SQLJ.REMOVE_JAR procedure that provide non-zero values of the `undeploy` parameter.

12.4 SQLJ.ALTER_JAVA_PATH procedure

Function

Alter the SQL-Java path of a JAR.

Signature

```
SQLJ.ALTER_JAVA_PATH (
    jar      IN    CHARACTER VARYING (L),
    path     IN    CHARACTER VARYING (L) )
```

Where *L* is an implementation-defined (IL015) integer value.

Syntax Rules

None.

Access Rules

1) The privileges required to invoke the SQLJ.ALTER_JAVA_PATH procedure are implementation-defined (IW206).

2) The current user shall be the owner of the JAR specified by the value of the jar parameter.

3) The current user shall have the USAGE privilege on each JAR referenced in the path parameter.

General Rules

1) The SQLJ.ALTER_JAVA_PATH procedure is subject to implementation-defined (IA152) rules for executing SQL-schema statements within SQL-transactions.

2) Let *J* be the value of the jar parameter. Let *TJ* be the value of

```
TRIM ( BOTH ' ' FROM J )
```

If *TJ* does not conform to the format of <jar name>, then an exception condition is raised: *Java-related condition — invalid JAR name (46002)*.

3) Let *JN* be the explicitly or implicitly qualified <jar id> specified in the <jar name> specified by *TJ*.

4) Let *JR* be the JAR identified by *JN*.

5) Let *P* be the value of the path parameter. If *P* does not conform to the format for <SQL Java path>, then an exception condition is raised: *Java-related condition — invalid path (4600D)*.

NOTE 72 — The path parameter can be an empty or all-blank string.

6) The current catalog and schema at the time of the call to the SQLJ.ALTER_JAVA_PATH procedure are the default, respectively, for each omitted <catalog name> and <schema name> in the <resolution jar>s contained in *P*. For each <path element> *PE* (if any) in *P*, *PE*'s <resolution jar> is updated to reflect the defaults for any omitted <catalog name>s and <schema name>s.

7) For each <path element> *PE* (if any) in *P*, let *RJ* be the <jar name> contained in the <resolution jar> contained in *PE*. If *RJ* is equivalent to *JN*, then an exception condition is raised: *Java-related condition — self-referencing path (4600E)*.

8) If *P* cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning — SQL-Java path too long for information schema (01011)*.

   NOTE 73 — The Information Schema is defined in ISO/IEC 9075-11.

9) The value of *P* is placed in the SQL-Java path of the JAR descriptor of *JR*, replacing the current SQL-Java path (if any) associated with *JR*.

10) If an invocation of the `SQLJ.ALTER_JAVA_PATH` procedure raises an exception condition, then effect on the path associated with the JAR is implementation-defined (IA153).

11) The effect of `SQLJ.ALTER_JAVA_PATH` on SQL statements that have already been prepared or are currently executing is implementation-dependent (UA009).

Conformance Rules

1) Without Feature J601, “SQL-Java paths”, conforming SQL language shall not contain invocations of the `SQLJ.ALTER_JAVA_PATH` procedure.

13 Java topics

13.1 Java facilities supported by this document

13.1.1 Package java.sql

SQL systems that implement this document support the package `java.sql`, which is the JDBC driver, and all classes required by that package. The other Java packages supplied by SQL systems that implement this document are implementation-defined (IE004).

In an SQL system that implements this document, the package `java.sql` supports the *default connection*. The default connection for a Java method invoked as an SQL routine has the following characteristics.

— The default connection is pre-allocated to provide efficient access to the database.

— The default connection is included in the current session and transaction.

— The authorization ID of the default connection is the current authorization ID.

— The JDBC AUTOCOMMIT setting of the default connection is *false*.

Other data source URLs that are supported by `java.sql` are implementation-defined (IA154).

13.1.2 System properties

SQL systems that implement this document support the system properties specified in Table 2, "System properties", for use by the `getProperty` method of `java.lang.System`:

Table 2 — System properties

Key	**Description of associated value**
`sqlj.defaultconnection`	If a Java method is executing in an SQL-implementation, then the String `"jdbc:default:connection"`[1]
`sqlj.runtime`	The class name of a runtime context class[2]

[1] Otherwise, the null value.

[2] This class is a subclass of the class `sqlj.runtime.RuntimeContext`. The `getDefaultContext()` method of the class whose name is returned returns the default connection described in Subclause 13.1.1, "Package java.sql".

13.2 Deployment descriptor files

Function

Supply information for actions to be taken by the `SQLJ.INSTALL_JAR` and `SQLJ.REMOVE_JAR` procedures.

Model

A deployment descriptor file is a text file contained in a JAR, which is specified with the following property in the manifest for the JAR:

```
Name: file_name
SQLJDeploymentDescriptor: TRUE
```

Properties

The text contained in a deployment descriptor file shall have the following form:

```
<descriptor file> ::=
  SQLActions <left bracket> <right bracket> <equals operator>
      { [ <double quote> <action group> <double quote>
      [ <comma> <double quote> <action group> <double quote> ] ] }

<action group> ::=
    <install actions>
  | <remove actions>

<install actions> ::=
  BEGIN INSTALL [ <command> <semicolon> ]... END INSTALL

<remove actions> ::=
  BEGIN REMOVE [ <command> <semicolon> ]... END REMOVE

<command> ::=
    <SQL statement>
  | <implementor block>

<SQL statement> ::=
  !! See Description

<implementor block> ::=
  BEGIN <implementor name> <SQL token>... END <implementor name>

<implementor name> ::=
  <identifier>

<SQL token> ::=
  !! See Description
```

Description

1) <descriptor file> shall contain at most one <install actions> and at most one <remove actions>.

2) The <command>s specified in the <install actions> (if any) and <remove actions> (if any) specify the *install actions* and *remove actions* of the deployment descriptor file, respectively.

3) An <SQL statement> specified in an <install actions> shall be either:

a) An <SQL-invoked routine> whose <language clause> specifies JAVA. The procedures and functions created by those statements are called the *deployed routines* of the deployment descriptor file.

b) A <grant privilege statement> that specifies the EXECUTE privilege for a deployed routine.

c) A <user-defined type definition> that specifies an <external Java type clause>. The types created by those statements are called the *deployed types* of the deployment descriptor file.

d) A <grant privilege statement> that specifies the USAGE privilege for a deployed type.

e) A <user-defined ordering definition> that specifies ordering for a deployed type.

4) When a deployment descriptor file is executed by a call of the `SQLJ.INSTALL_JAR` procedure, if the <jar name> of any <external routine name> or an <SQL-invoked routine> in an <install actions> is the <jar name> "`thisjar`", then "`thisjar`" is effectively replaced with the `jar` parameter of the `SQLJ.INSTALL_JAR` procedure for purposes of that execution.

5) An <SQL statement> specified in a <remove actions> shall be either:

a) A <drop routine statement> for a deployed routine.

b) A <revoke statement> for the EXECUTE privilege on a deployed routine.

c) A <drop data type statement> for a deployed type.

d) A <revoke statement> for the USAGE privilege on a deployed type.

e) A <drop user-defined ordering statement> that specifies ordering for a deployed type.

6) An <implementor block> specifies implementation-defined (IV112) install actions (remove actions) when specified in an <install actions> (<remove actions>).

7) An <SQL token> is an SQL lexical unit specified by the term "<token>" in Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2. That is, the comments, quotes, and double-quotes in an <implementor block> follow SQL token conventions.

8) An <implementor name> is an implementation-defined (IV208) SQL identifier. The <implementor name>s specified following the BEGIN and END keywords shall be equivalent.

9) Whether an <implementor block> with a given <implementor name> contained in an <install actions> (<remove actions>) is interpreted as an install action (remove action) is implementation-defined (IA155). That is, an SQL/JRT implementation may (but is not required to) perform install or remove actions specified by some other SQL/JRT implementation.

NOTE 74 — The deployment descriptor file format corresponds to the more general notion of a properties file supporting indexed properties. Therefore, the deployment descriptor file can be used by the SQL-implementation to instantiate a Java Bean, with an indexed property, `SQLActions`. An application developer could then customize the resulting Java Bean instance with ordinary Java Bean tools. For example, SQL procedures or permissions could be changed by changing the routine descriptors stored in the `SQLActions` property. The SQL system can then use the customized Java Bean instance to generate a modified version of the deployment descriptor file to use in a revised version of the JAR. Further information regarding Java Beans can be found in *The JavaBeans™ 1.01 Specification*, `https://download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/`.

Conformance Rules

1) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <user-defined type definition>.

2) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <drop data type statement>.

3) Without Feature J531, “Deployment”, conforming SQL language shall not contain an <SQL statement> that contains an <SQL-invoked routine>.

4) Without Feature J531, “Deployment”, conforming SQL language shall not contain an <SQL statement> that contains a <drop routine statement>.

5) Without Feature J531, “Deployment”, conforming SQL language shall not contain an <SQL statement> that contains a <user-defined ordering definition>.

6) Without Feature J531, “Deployment”, conforming SQL language shall not contain an <SQL statement> that contains a <drop user-defined ordering statement>.

7) Without Feature J531, “Deployment”, conforming SQL language shall not contain an <SQL statement> that contains a <grant privilege statement>.

8) Without Feature J531, “Deployment”, conforming SQL language shall not contain an <SQL statement> that contains a <revoke statement>.

14 Information Schema

This Clause modifies Clause 6, “Information Schema”, in ISO/IEC 9075-11.

14.1 Information Schema digital artifact

This Subclause modifies Subclause 6.1, “Information Schema digital artifact”, in ISO/IEC 9075-11.

Insert after the 1st paragraph: These schema definition and manipulation statements are also available from the ISO website as a “digital artifact”. See `https://standards.iso.org/iso-iec/9075/-13/ed-5/en/` to download digital artifacts for this document. To download the schema definition and manipulation statements, select the file named `ISO_IEC_9075-13(E)_JRT-schema-definition.sql`.

14.2 JAR_JAR_USAGE view

Function

Identify each JAR owned by a given user or role on which JARs defined in this catalog are dependent.

Definition

```
CREATE VIEW JAR_JAR_USAGE AS
    SELECT JJU.PATH_JAR_CATALOG, JJU.PATH_JAR_SCHEMA, JJU.PATH_JAR_NAME,
           JAR_CATALOG, JAR_SCHEMA, JAR_NAME
    FROM ( DEFINITION_SCHEMA.JAR_JAR_USAGE AS JJU
        JOIN
           DEFINITION_SCHEMA.JARS AS J
        USING ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME ) )
      JOIN
         DEFINITION_SCHEMA.SCHEMATA AS S
      ON ( ( JJU.PATH_JAR_CATALOG, JJU.PATH_JAR_SCHEMA )
         = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
          OR
            S.SCHEMA_OWNER IN
            ( SELECT ER.ROLE_NAME
              FROM ENABLED_ROLES AS ER ) )
       AND
          JAR_CATALOG =
          ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME AS ISCN ) ;

GRANT SELECT ON TABLE JAR_JAR_USAGE
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature J652, “SQL/JRT Usage tables”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.JAR_JAR_USAGE.

14.3 JARS view

Function

Identify the installed JARs defined in this catalog that are accessible to the current user.

Definition

```
CREATE VIEW JARS AS
    SELECT JAR_CATALOG, JAR_SCHEMA, JAR_NAME, JAVA_PATH
    FROM DEFINITION_SCHEMA.JARS
    WHERE ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME, 'JAR' ) IN
          ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
            FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
            WHERE GRANTEE IN
                  ( 'PUBLIC', CURRENT_USER )
                OR
                  GRANTEE IN
                  ( SELECT ROLE_NAME
                    FROM ENABLED_ROLES ) )
       AND
          JAR_CATALOG =
          ( SELECT CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME ) ;

GRANT SELECT ON TABLE JARS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.JARS.

14.4 METHOD_SPECIFICATIONS view

This Subclause modifies Subclause 6.37, “METHOD_SPECIFICATIONS view”, in ISO/IEC 9075-11.

Function

Identify the methods in the catalog that are accessible to a given user or role.

Definition

Insert into the Definition after

```
           D.DECLARED_DATA_TYPE, D.DECLARED_NUMERIC_PRECISION,
           D.DECLARED_NUMERIC_SCALE
```

the code:

```
, EXTERNAL_NAME, IS_FIELD
```

Conformance Rules

1) Insert after the last CR: Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference the following columns in the view INFORMATION_SCHEMA.METHOD_SPECIFICATIONS:

   a) EXTERNAL_NAME.

   b) IS_FIELD.

14.5 ROUTINE_JAR_USAGE view

Function

Identify the JARs owned by a given user or role on which external Java routines defined in this catalog are dependent.

Definition

```
CREATE VIEW ROUTINE_JAR_USAGE AS
    SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
           RJU.JAR_CATALOG, RJU.JAR_SCHEMA, RJU.JAR_NAME
    FROM ( DEFINITION_SCHEMA.ROUTINE_JAR_USAGE AS RJU
        JOIN
           DEFINITION_SCHEMA.ROUTINES AS R
        USING ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME ) )
      JOIN
         DEFINITION_SCHEMA.SCHEMATA AS S
      ON ( ( RJU.JAR_CATALOG, RJU.JAR_SCHEMA ) =
           ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
          OR
            S.SCHEMA_OWNER IN
            ( SELECT ER.ROLE_NAME
              FROM ENABLED_ROLES AS ER ) )
       AND
          SPECIFIC_CATALOG =
          ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME AS ISCN ) ;

GRANT SELECT ON TABLE ROUTINE_JAR_USAGE
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature J652, “SQL/JRT Usage tables”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.ROUTINE_JAR_USAGE.

14.6 TYPE_JAR_USAGE view

Function

Identify the JARs owned by a given user or role on which external Java types defined in this catalog are dependent.

Definition

```
CREATE VIEW TYPE_JAR_USAGE AS
    SELECT USER_DEFINED_TYPE_CATALOG AS UDT_CATALOG,
           USER_DEFINED_TYPE_SCHEMA AS UDT_SCHEMA,
           USER_DEFINED_TYPE_NAME AS UDT_NAME,
           TJU.JAR_CATALOG, TJU.JAR_SCHEMA, TJU.JAR_NAME
    FROM ( DEFINITION_SCHEMA.TYPE_JAR_USAGE AS TJU
        JOIN
           DEFINITION_SCHEMA.USER_DEFINED_TYPES AS UDT
        USING ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
                USER_DEFINED_TYPE_NAME ) )
      JOIN
         DEFINITION_SCHEMA.SCHEMATA AS S
      ON ( ( TJU.JAR_CATALOG, TJU.JAR_SCHEMA ) =
           ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
          OR
            S.SCHEMA_OWNER IN
            ( SELECT ER.ROLE_NAME
              FROM ENABLED_ROLES AS ER ) )
       AND
          USER_DEFINED_TYPE_CATALOG =
          ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME AS ISCN ) ;

GRANT SELECT ON TABLE TYPE_JAR_USAGE
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature J652, "SQL/JRT Usage tables", conforming SQL language shall not reference the view INFORMATION_SCHEMA.TYPE_JAR_USAGE.

14.7 USER_DEFINED_TYPES view

This Subclause modifies Subclause 6.77, “USER_DEFINED_TYPES view”, in ISO/IEC 9075-11.

Function

Identify the user-defined types defined in this catalog that are accessible to a given user or role.

Definition

Insert into the Definition after

```
          DTD.DECLARED_DATA_TYPE, DTD.DECLARED_NUMERIC_PRECISION,
          DTD.DECLARED_NUMERIC_SCALE, DTD.MAXIMUM_CARDINALITY
```

the code:

```
, EXTERNAL_NAME, EXTERNAL_LANGUAGE, JAVA_INTERFACE
```

Conformance Rules

1) Insert after the last CR: Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference the following columns in the view INFORMATION_SCHEMA.USER_DEFINED_TYPES:

   a) EXTERNAL_NAME.

   b) EXTERNAL_LANGUAGE.

   c) JAVA_INTERFACE.

14.8 Short name views

This Subclause modifies Subclause 6.83, "Short name views", in ISO/IEC 9075-11.

Function

Provide alternative views that use only identifiers that do not require Feature F391, "Long identifiers".

Definition

Insert into the Definition before

```
                          ) AS
    SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
```

the code:

```
          , EXTERNAL_NAME,      IS_FIELD
```

Insert into the Definition before

```
    FROM INFORMATION_SCHEMA.METHOD_SPECIFICATIONS;
```

the code:

```
, IS_FIELD, CREATED
```

Insert into the Definition before

```
                                                               ) AS
    SELECT USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME,
```

the code:

```
          , EXTERNAL_NAME, EXTERNAL_LANGUAGE,  JAVA_INTERFACE
```

Insert into the Definition before

```
    FROM INFORMATION_SCHEMA.USER_DEFINED_TYPES;
```

the code:

```
          , EXTERNAL_NAME, EXTERNAL_LANGUAGE, JAVA_INTERFACE
```

Conformance Rules

1) Insert after the last CR: Without Feature J651, "SQL/JRT Information Schema", conforming SQL language shall not reference the following columns in the view INFORMATION_SCHEMA.METHOD_SPECS:
   a) EXTERNAL_NAME.
   b) IS_FIELD.
2) Insert after the last CR: Without Feature J651, "SQL/JRT Information Schema", conforming SQL language shall not reference the following columns in the view INFORMATION_SCHEMA.UDT_S:
   a) EXTERNAL_NAME.
   b) EXTERNAL_LANGUAGE.
   c) JAVA_INTERFACE.

15 Definition Schema

This Clause modifies Clause 7, "Definition Schema", in ISO/IEC 9075-11.

15.1 Definition Schema digital artifact

This Subclause modifies Subclause 7.1, "Definition Schema digital artifact", in ISO/IEC 9075-11.

Insert after the 1st paragraph: These schema definition and manipulation statements are also available from the ISO website as a "digital artifact". See https://standards.iso.org/iso-iec/9075/-13/ed-5/en/ to download digital artifacts for this document. To download the schema definition and manipulation statements, select the file named ISO_IEC_9075-13(E)_JRT-schema-definition.sql.

15.2 JAR_JAR_USAGE base table

Function

The JAR_JAR_USAGE table has one row for each JAR included in the SQL-Java path of a JAR.

Definition

```
CREATE TABLE JAR_JAR_USAGE (
    JAR_CATALOG                                 INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_SCHEMA                                  INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_NAME                                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PATH_JAR_CATALOG                            INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PATH_JAR_SCHEMA                             INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PATH_JAR_NAME                               INFORMATION_SCHEMA.SQL_IDENTIFIER,

    CONSTRAINT JAR_JAR_USAGE_PRIMARY_KEY
      PRIMARY KEY ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME,
                    PATH_JAR_CATALOG, PATH_JAR_SCHEMA, PATH_JAR_NAME ),

    CONSTRAINT JAR_JAR_USAGE_CHECK_REFERENCES_JARS
      CHECK ( PATH_JAR_CATALOG NOT IN
              ( SELECT CATALOG_NAME
                FROM SCHEMATA )
            OR
              ( PATH_JAR_CATALOG, PATH_JAR_SCHEMA, PATH_JAR_NAME ) IN
              ( SELECT JAR_CATALOG, JAR_SCHEMA, JAR_NAME
                FROM JARS ) ),

    CONSTRAINT JAR_JAR_USAGE_FOREIGN_KEY_JARS
      FOREIGN KEY ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME )
        REFERENCES JARS
    );
```

Description

1) The JAR_JAR_USAGE table has one row for each JAR *JP* identified by a <jar name> contained in an <SQL Java path> associated with JAR *J*.

2) The values of JAR_CATALOG, JAR_SCHEMA, and JAR_NAME are the <catalog name>, <unqualified schema name>, and <jar id>, respectively, of the <jar name> of the JAR (*J*) being described.

3) The values of PATH_JAR_CATALOG, PATH_JAR _SCHEMA, and PATH_JAR _NAME are the <catalog name>, <unqualified schema name>, and <jar id>, respectively, of the <jar name> of a JAR (*JP*) that is in the <SQL Java path> associated with JAR *J*.

Initial Table Population

None.

15.3 JARS base table

Function

The JARS table has one row for each installed JAR.

Definition

```
CREATE TABLE JARS (
    JAR_CATALOG                                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_SCHEMA                                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_NAME                                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAVA_PATH                                     INFORMATION_SCHEMA.CHARACTER_DATA,

    CONSTRAINT JARS_PRIMARY_KEY
      PRIMARY KEY ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME ),
    CONSTRAINT JAR_FOREIGN_KEY_SCHEMATA
      FOREIGN KEY ( JAR_CATALOG, JAR_SCHEMA )
        REFERENCES SCHEMATA
    );
```

Description

1) The values of JAR_CATALOG, JAR_SCHEMA, and JAR_NAME are the <catalog name>, <unqualified schema name>, and <jar id> of the <jar name> of the JAR being described.

2) Case

   a) If the character representation of the SQL-Java path in the descriptor of the JAR being described can be represented without truncation, then the value of JAVA_PATH is that character representation.

   b) Otherwise, the value of JAVA_PATH is the null value.

Initial Table Population

None.

15.4 METHOD_SPECIFICATIONS base table

This Subclause modifies Subclause 7.33, "METHOD_SPECIFICATIONS base table", in ISO/IEC 9075-11.

Function

The METHOD_SPECIFICATIONS base table has one row for each method specification.

Definition

Insert into the Definition after

```
    METHOD_LANGUAGE                                INFORMATION_SCHEMA.CHARACTER_DATA
      CONSTRAINT METHOD_SPECIFICATIONS_LANGUAGE_CHECK
        CHECK ( METHOD_LANGUAGE IN
                ( 'SQL', 'ADA', 'C',
                  'COBOL', 'FORTRAN', 'MUMPS', 'PASCAL', 'PLI'
```

the code:

```
, 'JAVA'
```

Insert into the Definition after

```
    RESULT_CAST_AS_LOCATOR                         INFORMATION_SCHEMA.YES_OR_NO,
```

the code:

```
    EXTERNAL_NAME                                  INFORMATION_SCHEMA.CHARACTER_DATA,
    IS_FIELD                                       INFORMATION_SCHEMA.CHARACTER_DATA

    CONSTRAINT METHOD_SPECIFICATIONS_IS_FIELD_CHECK
      CHECK ( IS_FIELD IN ( 'YES', 'NO' ) ),

    CONSTRAINT METHOD_SPECIFICATIONS_METHOD_COMBINATIONS
      CHECK ( ( ( METHOD_LANGUAGE = 'JAVA' )
             AND
                ( EXTERNAL_NAME, IS_FIELD ) IS NOT NULL )
            OR
              ( ( METHOD_LANGUAGE <> 'JAVA' )
             AND
                ( ( EXTERNAL_NAME, IS_FIELD )
                IS NULL ) ) ),

    CONSTRAINT METHOD_SPECIFICATIONS_FIELD_COMBINATIONS
      CHECK ( IS_FIELD = 'NO' OR IS_STATIC = 'YES' ),
```

Description

1) Insert after the last Description: Case:

   a) If the method being described is an external Java routine, then the value of EXTERNAL_NAME is the <Java method and parameter declarations> specified in the <external Java method clause> for that external Java data type.

   b) If the method being described is a static field of an external Java type, then the value of EXTERNAL_NAME is the <qualified Java field name> specified in the <static field method spec> of the method.

   c) Otherwise, the value of EXTERNAL_NAME is the null value.

2) Insert after the last Description: Case:

a) If the method being described is a static field of an external Java type, then the value of IS_FIELD is 'YES'.

b) If the method being described is an external Java type, then the value of IS_FIELD is 'NO'.

c) Otherwise, the value of IS_FIELD is the null value.

Initial Table Population

No additional Initial Table Population items.

15.5 ROUTINE_JAR_USAGE base table

Function

The ROUTINE_JAR_USAGE table has one row for each external Java routine that names a JAR in an <external Java reference string>.

Definition

```
CREATE TABLE ROUTINE_JAR_USAGE (
    SPECIFIC_CATALOG                                INFORMATION_SCHEMA.SQL_IDENTIFIER,
    SPECIFIC_SCHEMA                                 INFORMATION_SCHEMA.SQL_IDENTIFIER,
    SPECIFIC_NAME                                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_CATALOG                                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_SCHEMA                                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_NAME                                        INFORMATION_SCHEMA.SQL_IDENTIFIER,

    CONSTRAINT ROUTINE_JAR_USAGE_PRIMARY_KEY
      PRIMARY KEY ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
                    JAR_CATALOG, JAR_SCHEMA, JAR_NAME ),

    CONSTRAINT JAR_JAR_USAGE_CHECK_REFERENCES_JARS
      CHECK ( JAR_CATALOG NOT IN
              ( SELECT CATALOG_NAME
                FROM SCHEMATA )
            OR
              ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME ) IN
              ( SELECT JAR_CATALOG, JAR_SCHEMA, JAR_NAME
                FROM JARS ) ),

    CONSTRAINT JAR_JAR_USAGE_FOREIGN_KEY_ROUTINES
      FOREIGN KEY (SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME )
        REFERENCES ROUTINES
    );
```

Description

1) The ROUTINE_JAR_USAGE table has one row for each external Java routine that names a JAR in an <external Java reference string>.

2) The values of SPECIFIC_CATALOG, SPECIFIC_SCHEMA, and SPECIFIC_NAME are the <catalog name>, <unqualified schema name>, and <qualified identifier>, respectively, of the <specific name> of the external Java routine being described.

3) The values of JAR_CATALOG, JAR_SCHEMA, and JAR_NAME are the <catalog name>, <unqualified schema name>, and <jar id>, respectively, of the <jar name> of the JAR being referenced in the external Java routine's <external Java reference string>.

Initial Table Population

None.

15.6 ROUTINES base table

This Subclause modifies Subclause 7.46, "ROUTINES base table", in ISO/IEC 9075-11.

Function

The ROUTINES table has one row for each SQL-invoked routine.

Definition

Insert into the Definition after

```
          CHECK ( EXTERNAL_LANGUAGE IN
                  ( 'ADA', 'C', 'COBOL',
                    'FORTRAN', 'MUMPS', 'PASCAL', 'PLI'
```

the code:

```
, 'JAVA'
```

Description

No additional Descriptions.

Initial Table Population

No additional Initial Table Population items.

15.7 TYPE_JAR_USAGE base table

Function

The TYPE_JAR_USAGE table has one row for each external Java type.

Definition

```
CREATE TABLE TYPE_JAR_USAGE (
    USER_DEFINED_TYPE_CATALOG                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
    USER_DEFINED_TYPE_SCHEMA                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
    USER_DEFINED_TYPE_NAME                         INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_CATALOG                                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_SCHEMA                                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_NAME                                       INFORMATION_SCHEMA.SQL_IDENTIFIER,

    CONSTRAINT TYPE_JAR_USAGE_PRIMARY_KEY
      PRIMARY KEY (USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
                   USER_DEFINED_TYPE_NAME, JAR_CATALOG, JAR_SCHEMA, JAR_NAME),

    CONSTRAINT TYPE_JAR_USAGE_CHECK_REFERENCES_JARS
      CHECK ( JAR_CATALOG NOT IN
              ( SELECT CATALOG_NAME
                FROM SCHEMATA )
            OR
              ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME ) IN
              ( SELECT JAR_CATALOG, JAR_SCHEMA, JAR_NAME
                FROM JARS ) ),

    CONSTRAINT TYPE_JAR_USAGE_FOREIGN_KEY_USER_DEFINED_TYPES
      FOREIGN KEY (USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
                   USER_DEFINED_TYPE_NAME ) REFERENCES USER_DEFINED_TYPES
    );
```

Description

1) The TYPE_JAR_USAGE table has one row for each external Java type.

2) The values of USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, and USER_DEFINED_TYPE_NAME are the <catalog name>, <unqualified schema name>, and <qualified identifier>, respectively, of the <user-defined type name> of the external Java type being described.

3) The values of JAR_CATALOG, JAR_SCHEMA, and JAR_NAME are the <catalog name>, <unqualified schema name>, and <jar id>, respectively, of the <jar name> of the JAR being referenced in the external Java type's <external Java class clause>.

Initial Table Population

None.

15.8 USAGE_PRIVILEGES base table

This Subclause modifies Subclause 7.65, "USAGE_PRIVILEGES base table", in ISO/IEC 9075-11.

Function

The USAGE_PRIVILEGES table has one row for each usage privilege descriptor. It effectively contains a representation of the usage privilege descriptors.

Definition

Insert into the Definition after

```
    CONSTRAINT USAGE_PRIVILEGES_OBJECT_TYPE_CHECK
      CHECK ( OBJECT_TYPE IN
            ( 'DOMAIN', 'CHARACTER SET',
              'COLLATION', 'TRANSLATION', 'SEQUENCE'
```

the code:

```
, 'JAR'
```

Insert into the Definition after

```
              UNION
                SELECT SEQUENCE_CATALOG, SEQUENCE_SCHEMA, SEQUENCE_NAME,
                       'SEQUENCE'
                FROM SEQUENCES
```

the code:

```
    UNION
      SELECT JAR_CATALOG, JAR_SCHEMA, JAR_NAME, 'JAR'
      FROM JARS
```

Description

1) Insert into Description 4), after the last list item:

JAR	The object to which the privilege applies is a JAR.

Initial Table Population

No additional Initial Table Population items.

15.9 USER_DEFINED_TYPES base table

This Subclause modifies Subclause 7.67, "USER_DEFINED_TYPES base table", in ISO/IEC 9075-11.

Function

The USER_DEFINED_TYPES table has one row for each user-defined type.

Definition

Insert into the Definition after

```
      CONSTRAINT USER_DEFINED_TYPES_ORDERING_CATEGORY_CHECK
        CHECK ( ORDERING_CATEGORY IN
                ( 'RELATIVE', 'MAP', 'STATE'
```

the code:

```
, 'COMPARABLE'
```

Insert into the Definition after

```
    REF_DTD_IDENTIFIER                             INFORMATION_SCHEMA.SQL_IDENTIFIER,
```

the code:

```
    EXTERNAL_NAME                                  INFORMATION_SCHEMA.CHARACTER_DATA,
    EXTERNAL_LANGUAGE                              INFORMATION_SCHEMA.CHARACTER_DATA
      CONSTRAINT USER_DEFINED_TYPE_EXTERNAL_LANGUAGE_CHECK
        CHECK ( EXTERNAL_LANGUAGE IN ( 'JAVA' ) ),
    JAVA_INTERFACE                                 INFORMATION_SCHEMA.CHARACTER_DATA
      CONSTRAINT USER_DEFINED_TYPE_JAVA_INTERFACE_CHECK
        CHECK ( JAVA_INTERFACE IN ( 'SERIALIZABLE', 'SQLDATA' ) ),
    CONSTRAINT USER_DEFINED_TYPES_COMBINATIONS
      CHECK ( ( ( EXTERNAL_LANGUAGE = 'JAVA' ) AND
                ( EXTERNAL_NAME, JAVA_INTERFACE ) IS NOT NULL )
          OR
            ( ( EXTERNAL_LANGUAGE, EXTERNAL_NAME, JAVA_INTERFACE )
              IS NULL ) )
```

Description

1) Insert into Description 7), after the last list item:

   COMPARABLE Two values of this type may be compared with `java.lang.Comparable`'s `compareTo( )` method.

2) Insert after the last Description: Case:

   a) If the user-defined type being described is an external Java data type, then the value of EXTERNAL_NAME is the <jar and class name> specified in the <external Java class clause> for that external Java data type.

   b) Otherwise, the value of EXTERNAL_NAME is the null value.

3) Insert after the last Description: Case:

   a) If the user-defined type being described is an external Java data type, then the value of EXTERNAL_LANGUAGE is 'JAVA'.

   b) Otherwise, the value of EXTERNAL_LANGUAGE is the null value.