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)

Documents
Document Sharing Hub
© ISO/IEC 2023

Full document details: ISO IEC 09075-13-2023

**COPYRIGHT PROTECTED DOCUMENT**

© ISO/IEC 2023

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland



© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO IEC 09075-13-2023

# 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

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

**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


© ISO/IEC 2023 – All rights reserved
Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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**

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

# 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


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

# 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;

Documents
Document Sharing Hub

© ISO/IEC 2023 – All rights reserved


Full document details: ISO IEC 09075-13-2023

— 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.

Documents
Document Sharing Hub


© ISO/IEC 2023 – All rights reserved

Full document details: ISO IEC 09075-13-2023

# 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.

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub

Full document details: ISO IEC 09075-13-2023

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.


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

# 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.

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

## 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


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

# 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

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

**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`


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

**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)

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

# 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.


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

— **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

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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., `myclass.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.


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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.

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

## 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*.


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

*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.*

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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.


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

— 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.

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

- 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


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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.


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

# 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.*

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

## 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.


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

## 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.*

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

# 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.*


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

## 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.

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

# 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`


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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.*

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

# 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.


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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*.

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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.


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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 $ESP_i$.

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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 ) ;
       ```


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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 PD1 = new
JCLSN ( PD2, ...,
PDJPEN ) ;
```

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

```
JCLSN PD1 = 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 =
    PD1.JMN ( PD2, ..., PDJPEN ) ;
```

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

```
SJCEN tempU = PD1.JMN ( ) ;
```

NOTE 21 — The Java method effectively executed by either the Java statement `SJCEN tempU = PD1.JMN ( PD2, ..., PDJPEN ) ;` or the Java statement `SJCEN tempU = PD1.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*.

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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:


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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:

© ISO/IEC 2023 – All rights reserved

Documents
Document Sharing Hub
Full document details: ISO IEC 09075-13-2023

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.*


© ISO/IEC 2023 – All rights reserved
Documents Document Sharing Hub
Full document details: ISO IEC 09075-13-2023