INTERNATIONAL STANDARD

ISO/IEC 9075-13

Third edition
2008-07-15

Information technology — Database languages — SQL —

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

Reference number
ISO/IEC 9075-13:2008(E)

© ISO/IEC 2008

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2008

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

--`,,```,,,,````-`-`,,`,,`,`,,`---

::

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

© ISO/IEC 2008 – All rights reserved

Not for Resale

Contents

Page

Foreword. ix
Introduction. x
1 Scope. 1
2 Normative references. 3
2.1 ISO and IEC standards. 3
2.2 Other international standards. 3
3 Definitions, notations, and conventions. 5
3.1 Definitions. 5
3.1.1 Definitions taken from [Java]. 5
3.1.2 Definitions taken from [JVM]. 5
3.1.3 Definitions provided in Part 13. 6
3.2 Conventions. 6
3.2.1 Specification of built-in procedures. 7
3.2.2 Specification of deployment descriptor files. 7
4 Concepts. 9
4.1 The Java programming language. 9
4.2 SQL-invoked routines. 10
4.2.1 Overview of SQL-invoked routines. 10
4.2.2 Characteristics of SQL-invoked routines. 10
4.3 Java class name resolution. 12
4.4 SQL result sets. 13
4.5 Parameter mapping. 13
4.6 Unhandled Java exceptions. 14
4.7 Data types. 15
4.7.1 Host language data types. 15
4.8 User-defined types. 15
4.8.1 Introduction to user-defined types. 15
4.8.2 User-defined type descriptor. 16
4.8.3 User-defined type comparison and assignment. 18
4.8.4 Accessing static fields. 18
4.8.5 Converting objects between SQL and Java. 18
4.8.5.1 SERIALIZABLE. 19
4.8.5.2 SQLDATA. 19
4.8.5.3 Developing for portability. 20
4.9 Built-in procedures. 20

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.10 Privileges. . . . 21
4.11 JARs. . . . 21
4.11.1 Deployment descriptor files. . . . 22
5 Lexical elements. . . . 23
5.1 <token> and <separator>. . . . 23
5.2 Names and identifiers. . . . 25
6 Scalar expressions. . . . 27
6.1 <method invocation>. . . . 27
6.2 <new specification>. . . . 28
7 Predicates. . . . 29
7.1 <comparison predicate>. . . . 29
8 Additional common elements. . . . 31
8.1 <Java parameter declaration list>. . . . 31
8.2 <SQL Java path>. . . . 32
8.3 <routine invocation>. . . . 34
8.4 <language clause>. . . . 43
8.5 Execution of array-returning functions. . . . 44
8.6 Java routine signature determination. . . . 51
9 Schema definition and manipulation. . . . 59
9.1 <drop schema statement>. . . . 59
9.2 <table definition>. . . . 61
9.3 <view definition>. . . . 62
9.4 <user-defined type definition>. . . . 63
9.5 <attribute definition>. . . . 67
9.6 <alter type statement>. . . . 71
9.7 <drop data type statement>. . . . 72
9.8 <SQL-invoked routine>. . . . 73
9.9 <alter routine statement>. . . . 77
9.10 <drop routine statement>. . . . 78
9.11 <user-defined ordering definition>. . . . 79
9.12 <drop user-defined ordering statement>. . . . 80
10 Access control. . . . 81
10.1 <grant privilege statement>. . . . 81
10.2 <privileges>. . . . 82
10.3 <revoke statement>. . . . 83
11 Built-in procedures. . . . 85
11.1 SQLJ.INSTALL_JAR procedure. . . . 85
11.2 SQLJ.REPLACE_JAR procedure. . . . 87
11.3 SQLJ.REMOVE_JAR procedure. . . . 89
11.4 SQLJ.ALTER_JAVA_PATH procedure. . . . 91
12 Java topics. . . . 93

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12.1 Java facilities supported by this part of ISO/IEC 9075. . . . 93
12.1.1 Package java.sql. . . . 93
12.1.2 System properties. . . . 93
12.2 Deployment descriptor files. . . . 94
13 Information Schema. . . . 97
13.1 JAR_JAR_USAGE view. . . . 97
13.2 JARS view. . . . 98
13.3 METHOD_SPECIFICATIONS view. . . . 99
13.4 ROUTINE_JAR_USAGE view. . . . 100
13.5 TYPE_JAR_USAGE view. . . . 101
13.6 USER_DEFINED_TYPES view. . . . 102
13.7 Short name views. . . . 103
14 Definition Schema. . . . 105
14.1 JAR_JAR_USAGE base table. . . . 105
14.2 JARS base table. . . . 107
14.3 METHOD_SPECIFICATIONS base table. . . . 108
14.4 ROUTINE_JAR_USAGE base table. . . . 110
14.5 ROUTINES base table. . . . 111
14.6 TYPE_JAR_USAGE base table. . . . 112
14.7 USAGE_PRIVILEGES base table. . . . 113
14.8 USER_DEFINED_TYPES base table. . . . 114
15 Status codes. . . . 117
15.1 SQLSTATE. . . . 117
16 Conformance. . . . 119
16.1 Claims of conformance to SQL/JRT. . . . 119
16.2 Additional conformance requirements for SQL/JRT. . . . 119
16.3 Implied feature relationships of SQL/JRT. . . . 119
Annex A SQL Conformance Summary (informative). . . . **121**
Annex B Implementation-defined elements (informative). . . . **127**
Annex C Implementation-dependent elements (informative). . . . **131**
Annex D Deprecated features (infomative). . . . **133**
Annex E Incompatibilities with ISO/IEC 9075:2003 (informative). . . . **135**
Annex F SQL feature taxonomy (informative). . . . **137**
Annex G Defect reports not addressed in this edition of this part of ISO/IEC 9075 (informative). . . **139**
Annex H Routines tutorial (informative). . . . **141**
H.1 Technical components. . . . 141
H.2 Overview. . . . 142
H.3 Example Java methods: region and correctStates. . . . 143
H.4 Installing region and correctStates in SQL. . . . 143
H.5 Defining SQL names for region and correctStates. . . . 144

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

H.6 A Java method with output parameters: bestTwoEmps. . . . 146
H.7 A CREATE PROCEDURE best2 for bestTwoEmps. . . . 147
H.8 Calling the best2 procedure. . . . 148
H.9 A Java method returning a result set: orderedEmps. . . . 148
H.10 A CREATE PROCEDURE rankedEmps for orderedEmps. . . . 150
H.11 Calling the rankedEmps procedure. . . . 151
H.12 Overloading Java method names and SQL names. . . . 152
H.13 Java main methods. . . . 153
H.14 Java method signatures in the CREATE statements. . . . 154
H.15 Null argument values and the RETURNS NULL clause. . . . 155
H.16 Static variables. . . . 157
H.17 Dropping SQL names of Java methods. . . . 158
H.18 Removing Java classes from SQL. . . . 159
H.19 Replacing Java classes in SQL. . . . 159
H.20 Visibility. . . . 160
H.21 Exceptions. . . . 161
H.22 Deployment descriptors. . . . 162
H.23 Paths. . . . 164
H.24 Privileges. . . . 166
H.25 Information Schema. . . . 167

Annex I Types tutorial (informative). . . . **169**
I.1 Overview. . . . 169
I.2 Example Java classes. . . . 169
I.3 Installing Address and Address2Line in an SQL system. . . . 171
I.4 CREATE TYPE for Address and Address2Line. . . . 172
I.5 Multiple SQL types for a single Java class. . . . 173
I.6 Collapsing subclasses. . . . 174
I.7 GRANT and REVOKE statements for data types. . . . 176
I.8 Deployment descriptors for classes. . . . 176
I.9 Using Java classes as data types. . . . 177
I.10 SELECT, INSERT, and UPDATE. . . . 178
I.11 Referencing Java fields and methods in SQL. . . . 179
I.12 Extended visibility rules. . . . 179
I.13 Logical representation of Java instances in SQL. . . . 180
I.14 Static methods. . . . 181
I.15 Static fields. . . . 182
I.16 Instance-update methods. . . . 183
I.17 Subtypes in SQL/JRT data. . . . 184
I.18 References to fields and methods of null instances. . . . 185
I.19 Ordering of SQL/JRT data. . . . 187

Bibliography. . . . 189

Index. . . . 191

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Tables

Table

Page

1 Standard programming languages. 43
2 System properties. 93
3 SQLSTATE class and subclass values. 117
4 Implied feature relationships of SQL/JRT. 119
5 Feature taxonomy for optional features. 137

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 9075-13 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

This third edition of ISO/IEC 9075-13 cancels and replaces the second edition (ISO/IEC 9075-13:2003), which has been technically revised. It also incorporates Technical Corrigendum ISO/IEC 9075-13:2003/Cor.1:2005.

ISO/IEC 9075 consists of the following parts, under the general title *Information technology — Database languages — SQL*:

— Part 1: Framework (SQL/Framework)

— Part 2: Foundation (SQL/Foundation)

— Part 3: Call-Level Interface (SQL/CLI)

— Part 4: Persistent Stored Modules (SQL/PSM)

— Part 9: Management of External Data (SQL/MED)

— Part 10: Object Language Bindings (SQL/OLB)

— Part 11: Information and Definition Schema (SQL/Schemata)

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

— Part 14: XML-Related Specifications (SQL/XML)

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

The organization of this part of ISO/IEC 9075 is as follows:

1) Clause 1, "Scope", specifies the scope of this part of ISO/IEC 9075.

2) Clause 2, "Normative references", identifies additional standards that, through reference in this part of ISO/IEC 9075, constitute provisions of this part of ISO/IEC 9075.

3) Clause 3, "Definitions, notations, and conventions", defines the notations and conventions used in this part of ISO/IEC 9075.

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 elements", defines additional language elements that are used in various parts of the language.

9) Clause 9, "Schema definition and manipulation", defines the schema definition and manipulation statements associated with the definition of Java routines and types.

10) Clause 10, "Access control", defines facilities for controlling access to SQL-data.

11) Clause 11, "Built-in procedures", defines new built-in procedures used in the definition of Java routines and types.

12) Clause 12, "Java topics", defines the facilities supported by implementations of this part of ISO/IEC 9075 and the conventions used in deployment descriptor files.

13) Clause 13, "Information Schema", defines viewed tables that contain schema information.

14) Clause 14, "Definition Schema", defines base tables on which the viewed tables containing schema information depend.

15) Clause 15, "Status codes", defines SQLSTATE values related to Java routines and types.

16) Clause 16, "Conformance", defines the criteria for conformance to this part of ISO/IEC 9075.

17) Annex A, "SQL Conformance Summary", is an informative Annex. It summarizes the conformance requirements of the SQL language.

18) Annex B, "Implementation-defined elements", is an informative Annex. It lists those features for which the body of this part of ISO/IEC 9075 states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-defined.

19) Annex C, "Implementation-dependent elements", is an informative Annex. It lists those features for which the body of this part of ISO/IEC 9075 states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-dependent.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

20) Annex D, “Deprecated features”, is an informative Annex. It lists features that the responsible Technical Committee intend will not appear in a future revised version of this part of ISO/IEC 9075.

21) Annex E, “Incompatibilities with ISO/IEC 9075:2003”, is an informative Annex. It lists incompatibilities with the previous version of this part of ISO/IEC 9075.

22) Annex F, “SQL feature taxonomy”, is an informative Annex. It identifies features of the SQL language specified in this part of ISO/IEC 9075 by an identifier and a short descriptive name. This taxonomy is used to specify conformance and may be used to develop other profiles involving the SQL language.

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

24) Annex H, “Routines tutorial”, is an informative Annex. It provides a tutorial on using the features defined in this part of ISO/IEC 9075 for defining and using SQL-invoked routines based on Java static methods.

25) Annex I, “Types tutorial”, is an informative Annex. It provides a tutorial on using the features defined in this part of ISO/IEC 9075 for defining and using SQL structured types based on Java classes.

In the text of this part of ISO/IEC 9075, Clauses begin a new odd-numbered page, and in Clause 5, “Lexical elements”, through Clause 16, “Conformance”, Subclauses begin a new page. Any resulting blank space is not significant.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Information technology — Database languages — SQL —

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

1 Scope

This part of ISO/IEC 9075 specifies the ability to invoke static methods written in the Java™ programming language as SQL-invoked routines and to use classes defined in the Java programming language as SQL structured user-defined types. (Java is a registered trademark of Sun Microsystems, Inc.)

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

2 Normative references

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

2.1 ISO and IEC standards

[ISO9075-1] ISO/IEC 9075-1:2008, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*.

[ISO9075-2] ISO/IEC 9075-2:2008, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*.

[ISO9075-10] ISO/IEC 9075-10:2008, *Information technology — Database languages — SQL — Part 10: Object Language Bindings (SQL/OLB)*.

[ISO9075-11] ISO/IEC 9075-11:2008, *Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata)*.

2.2 Other international standards

[Java] *The Java™ Language Specification, Third Edition*, James Gosling, Bill Joy, Guy Steele, and Gilad Bracha, Prentice Hall, June 14, 2005, ISBN 0-321-24678-0.

[JVM] *The Java™ Virtual Machine Specification, Second Edition*, Tim Lindholm and Frank Yellin, Addison-Wesley, 1999, ISBN 0-201-43294-3, as amended by *Clarifications and Amendments to the Java Virtual Machine Specification*, `http://java.sun.com/docs/books/jvms/second_edition/jvms-clarify.html`.

[J2SE] *Java™ 2 Platform Standard Edition 5.0 API Specification*, `http://java.sun.com/j2se/1.5.0/docs/api/index.html`.

[Serialization] *Java™ Object Serialization Specification*, version 1.5.0 `http://java.sun.com/j2se/1.5.0/docs/guide/serialization/spec/serial-TOC.html`.

[JDBC] *JDBC™ 4.0 Specification*, Final v1.0, Lance Andersen, Sun Microsystems, Inc., November 7, 2006.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3 Definitions, notations, and conventions

This Clause modifies Clause 3, "Definitions, notations, and conventions", in ISO/IEC 9075-2.

3.1 Definitions

This Subclause modifies Subclause 3.1, "Definitions", in ISO/IEC 9075-2.

3.1.1 Definitions taken from [Java]

For the purposes of this document, the definitions of the following terms given in [Java] apply.

3.1.1.1 block

3.1.1.2 class declaration

3.1.1.3 class instance

3.1.1.4 class variable

3.1.1.5 field

3.1.1.6 instance initializer

3.1.1.7 instance variable

3.1.1.8 interface

3.1.1.9 local variable

3.1.1.10 nested class

3.1.1.11 package

3.1.1.12 static initializer

3.1.1.13 subpackage

3.1.2 Definitions taken from [JVM]

For the purposes of this document, the definitions of the following terms given in [JVM] apply.

3.1.2.1 class file

3.1.2.2 Java Virtual Machine

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.1.3 Definitions provided in Part 13

For the purposes of this document, in addition to those definitions adopted from other sources, the following definitions apply:

3.1.3.1 **default connection**: A JDBC connection to the current SQL-implementation, SQL-session, and SQL-transaction established with the data source URL `'jdbc:default:connection'`. (See [RFC2368] and [RFC3986] for more details about URLs.)

3.1.3.2 **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*.

3.1.3.3 **deployment descriptor file**: A text file containing deployment descriptors that is contained in a JAR, for which the JAR's manifest entry, as described by the `java.util.jar` section of [J2SE], specifies `SQLJDeploymentDescriptor: TRUE`.

3.1.3.4 **external Java data type**: An SQL user-defined type defined with a <user-defined type definition> that specifies an <external Java type clause>.

3.1.3.5 **external Java routine**: An 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>.

3.1.3.6 **installed JAR**: A JAR whose existence has been registered with the SQL-environment and whose contents have been copied into that SQL-environment due to execution of one of the procedures `SQLJ.INSTALL_JAR` and `SQLJ.REPLACE_JAR`.

3.1.3.7 **Java Archive (JAR)**: A zip formatted file, as described by the `java.util.zip` section of [J2SE], containing zero or more Java `class` and `ser` files, and zero or more deployment descriptor files. JARs are a normal vehicle for distributing Java programs and the mechanism specified by this International Standard to provide the implementation of external Java routines and external Java data types to an SQL-environment.

3.1.3.8 **JVM**: A Java Virtual Machine, as defined by [JVM].

3.1.3.9 **ser file**: A file containing representations of Java objects in the form defined in [Serialization].

3.1.3.10 **subject Java class**: The Java class uniquely identified by the combination of the class's *subject Java class name* and its containing JAR.

3.1.3.11 **subject Java class name**: The fully-qualified package and class name of a Java class.

3.1.3.12 **system class**: Any Java class provided by a conforming implementation of this part of ISO/IEC 9075 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.

3.2 Conventions

This Subclause modifies Subclause 3.3, "Conventions", in ISO/IEC 9075-2.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.2.1 Specification of built-in procedures

Built-in procedures are specified in terms of:

— **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.

— **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.

3.2.2 Specification of deployment descriptor files

Deployment descriptor files are specified in terms of:

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

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

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

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

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

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4 Concepts

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

4.1 The Java programming language

The Java programming language is a class-based, object-oriented language. This part of ISO/IEC 9075 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 name space and classes within a package are

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

able to refer to other classes in the same package, including classes not referenceable from outside the package. Every class belongs to exactly one package, either an explicitly specified named package or the anonymous default package. A class can specify Java `import` statements to refer to other named packages whose classes can then be referenced within the class without package qualification.

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

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

The Java *Serializable* interface, `java.io.Serializable`, as described in [J2SE], 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 [J2SE] 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 [J2SE], 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.

4.2 SQL-invoked routines

This Subclause modifies Subclause 4.27, "SQL-invoked routines", in ISO/IEC 9075-2.

4.2.1 Overview of SQL-invoked routines

This Subclause modifies Subclause 4.27.1, "Overview of SQL-invoked routines", in ISO/IEC 9075-2.

Replace the lead text of the 9th paragraph A static SQL-invoked method, whether or not it is an external Java routine, satisfies the following conditions:

4.2.2 Characteristics of SQL-invoked routines

This Subclause modifies Subclause 4.27.2, "Characteristics of SQL-invoked routines", in ISO/IEC 9075-2.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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:

— 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 8.2, "<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.

Replace the 4th paragraph SQL-invoked routines are invoked differently depending on their form. SQL-invoked procedures are invoked by <call statement>s. SQL-invoked regular functions are invoked by <routine invocation>s. Instance SQL-invoked methods are invoked by <method invocation>s, while SQL-invoked constructor methods are invoked by <new invocation>s and static SQL-invoked methods are invoked by <static method invocation>s. An invocation of an SQL-invoked routine specifies the <routine name> of the SQL-invoked routine and supplies a sequence of argument values corresponding to the <SQL parameter declaration>s of the SQL-invoked routine. A *subject routine* of an invocation is an SQL-invoked routine that may be invoked by a <routine invocation>. After the selection of the subject routine of a <routine invocation>, the SQL arguments are evaluated and the SQL-invoked routine that will be executed is selected. If the subject routine is an instance SQL-invoked method that is not an external Java routine, then the SQL-invoked routine that is executed is selected from the set of overriding methods of the subject routine. (The term "set of overriding methods" is defined in the General Rules of Subclause 10.4, "<routine invocation>", in [ISO9075-2].) The overriding method that is selected is the overriding method with a subject parameter the type designator of whose declared type precedes that of the declared type of the subject parameter of every other overriding method in the type precedence list of the most specific type of the value of the SQL argument that corresponds to the subject parameter. See the General Rules of Subclause 10.4, "<routine invocation>", in [ISO9075-2]. 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. If the subject routine is not an SQL-invoked method, then the SQL-invoked routine executed is that subject routine. After the selection of the SQL-invoked routine for execution, the values of the SQL arguments are assigned to the corresponding SQL parameters of the SQL-invoked routine and its <routine body> is executed. If the SQL-invoked routine is an SQL routine, then the <routine body> is an <SQL procedure statement> that is executed according to the General Rules of Subclause 13.5, "<SQL procedure statement>", in [ISO9075-2]. If the SQL-invoked routine is an external routine, then the <routine body> identifies a program written in some programming language other than SQL that is executed according to the rules of that programming language.

Replace the 6th paragraph If the SQL-invoked routine is an external routine, then an effective SQL parameter list is constructed before the execution of the <routine body>. The effective SQL parameter list has different entries depending on the parameter passing style of the SQL-invoked routine. The value of each entry in the effective SQL parameter list is set according to the General Rules of Subclause 8.3, "<routine invocation>". When the SQL-invoked routine is not an external Java routine, the values in the effective SQL parameter list are passed to the program identified by the <routine body> according to the rules of Subclause 13.6, "Data type correspondences", in [ISO9075-2]; 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

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Subclause 4.5, "Parameter mapping". After the execution of that program, if the parameter passing style of the SQL-invoked routine is SQL, then the SQL-implementation obtains the values for output parameters (if any), the value (if any) returned from the program, the value of the exception data item, and the value of the message text (if any) from the values assigned by the program to the effective SQL parameter list. If the parameter passing style of the SQL-invoked routine is JAVA, then such values are obtained from the values assigned by the program to the effective SQL parameter list and the uncaught Java exception (if any). If the parameter passing style of the SQL-invoked routine is GENERAL, then such values are obtained in an implementation-defined manner.

4.3 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*):

— 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 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.4 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 [J2SE], 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 [ISO9075-2], 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 part of ISO/IEC 9075's treatment of result set cursors differs from that of [ISO9075-2].

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 part of ISO/IEC 9075, 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 4.5, "Parameter mapping", and Subclause 8.3, "<routine invocation>", 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 8.3, "<routine invocation>", the calling application will be returned zero or more dynamic result sets in the order in which the cursors were opened, just as in [ISO9075-2]; 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.

4.5 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:

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

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

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

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

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

A Java class is *result set oriented* if and only if it is either:

— 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 [ISO9075-10].

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*.

4.6 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.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

4.7 Data types

This Subclause modifies Subclause 4.1, "Data types", in ISO/IEC 9075-2.

4.7.1 Host language data types

This Subclause modifies Subclause 4.1.3, "Host language data types", in ISO/IEC 9075-2.

Replace the 1st paragraph Each host language has its own data types, which are separate and distinct from SQL data types, even though similar names may be used to describe the data types. Mappings of SQL data types to data types in host languages are described in Subclause 11.51, "<SQL-invoked routine>", in [ISO9075-2], in Subclause 21.1, "<embedded SQL host program>", in [ISO9075-2], and in Subclause 8.1, "<embedded SQL host program>", in [ISO9075-10]. Not every SQL data type has a corresponding data type in every host language.

4.8 User-defined types

This Subclause modifies Subclause 4.7, "User-defined types", in ISO/IEC 9075-2.

4.8.1 Introduction to user-defined types

This Subclause modifies Subclause 4.7.1, "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:

— 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.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— 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 9.4, "<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 ofthe subject Java class.

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

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

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

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

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

4.8.2 User-defined type descriptor

This Subclause modifies Subclause 4.7.2, "User-defined type descriptor", in ISO/IEC 9075-2.

Augment 1st paragraph | add the keyword COMPARABLE to the list in 4th list item

— Augment 1st paragraph | insert following 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 <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.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

  • The <method name>.

  • The <specific method name>.

  • The <SQL parameter declaration list>.

  • The <returns data type>, and indication of SELF AS RESULT.

  • The <result cast from type>, if any.

  • The package, class, and name of the Java routine corresponding to this method and, if specified, its signature.

  • An indication of whether STATIC or CONSTRUCTOR is specified.

  • If STATIC is specified, then an indication of whether this is a static field method.

  • If this is a static field method, then the <Java field name> of the static field and the <Java class name> of the class that declares that static field.

  • An indication of whether the method is deterministic.

  • An indication of whether the method possibly writes SQL data, possibly reads SQL data, possibly contains SQL, or does not possibly contain SQL.

  • An indication of whether the method should not 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:

— 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>.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- 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 should not 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>.

4.8.3 User-defined type comparison and assignment

This Subclause modifies Subclause 4.7.6, "User-defined type comparison and assignment", in ISO/IEC 9075-2.

Replace the 5th paragraph Let *comparison function* of a user-defined type T_a be the ordering function included in the user-defined type descriptor of the comparison type of T_a, if any.

Replace the 6th paragraph Two values *V1* and *V2* whose most specific types are user-defined types *T1* and *T2* are comparable if and only if *T1* and *T2* are in the same subtype family and each have some comparison type *CT1* and *CT2*, respectively. *CT1* and *CT2* constrain the comparison forms and comparison categories of *T1* and *T2* to be the same and to be the same as those of all their supertypes. If the comparison category is COMPARABLE, then no comparison functions shall be specified for *T1* and *T2*. If the comparison category is either STATE or RELATIVE, then the comparison functions of *T1* and *T2* are constrained to be equivalent. If the comparison category is MAP, they are not constrained to be equivalent.

4.8.4 Accessing static fields

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

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

4.8.5 Converting 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

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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 International Standard supports these 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 [J2SE] is used for object state conversion.

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

4.8.5.1 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).

4.8.5.2 SQLDATA

If the <interface specification> of a <user-defined type definition> specifies SQLDATA,then object state communicationis based on the Java interface `java.sql.SQLData` defined in [JDBC] and [J2SE]. 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 part of ISO/IEC 9075. The Java object representation may be entirely different from the SQL object attributes, if desired. For example, an SQL Point type may define a geometric point in terms of cartesian coordinates, while the corresponding Java class defines it using polar coordinates. The only requirement to be met

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

4.8.5.3 Developing for portability

The following guidelines provide maximum portability of Java classes across different implementations of this part of ISO/IEC 9075 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.

4.9 Built-in procedures

This part of ISO/IEC 9075 differs slightly from other parts of ISO/IEC 9075 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 International Standard introduces a set offour 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 part of ISO/IEC 9075, and to contain all of the built-in procedures defined in this part of ISO/IEC 9075.

Built-in procedures defined in this part of ISO/IEC 9075 are:

— `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.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.10 Privileges

This Subclause modifies Subclause 4.34.2, "Privileges", in ISO/IEC 9075-2.

Augment the list in the 1st paragraph

— JAR

Augment the list in the 8th paragraph

— JAR

Insert after the 8th paragraph The privileges for facilities defined in this part of ISO/IEC 9075 are as follows:

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

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 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.

4.11 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 catalog name, schema name, and JAR identifier of the JAR.

— The SQL-Java path of the JAR.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4.11.1 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 implementation. To assist the automation of repeated installations, deployment descriptor files contain the variants of SQL-schema statements defined in this part of ISO/IEC 9075. 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 implementation-defined *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 part of ISO/IEC 9075 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 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

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

  | INTERFACE

  | JAVA

  | SQLDATA

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

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance Rules

No additional Conformance Rules.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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 this 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 this SR The character set supported, and the maximum lengths of the <package identifier>, <class identifier>, <Java field name>, and <Java method name>are implementation-defined.

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 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Access Rules

No additional Access Rules.

General Rules

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

Conformance Rules

No additional Conformance Rules.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6 Scalar expressions

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

6.1 <method invocation>

This Subclause modifies Subclause 6.16, "<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 2) 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 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2 <new specification>

This Subclause modifies Subclause 6.18, "<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 this 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.

Conformance Rules

1) Insert this 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 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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) NOTE 7 — Replace Note 206 The comparison form and comparison categories included in the user-defined type descriptors of both *UDT1* and *UDT2* are constrained to be the same and to be the same as those of all their supertypes. If the comparison category is COMPARABLE, then no comparison functions are specified for *T1* and *T2*. If the comparison category is either STATE or RELATIVE, then *UDT1* and *UDT2* are constrained to have the same comparison function; if the comparison category is MAP, they are not constrained to have the same comparison function.

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 8 — Syntax Rules in Subclause 9.11, "<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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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

d) *X* < *Y*

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

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

e) *X* <> *Y*

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

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

f) *X* > *Y*

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

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

g) *X* <= *Y*

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

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

h) *X* >= *Y*

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

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

Conformance Rules

No additional Conformance Rules.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 Additional common elements

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

8.1 <Java parameter declaration list>

Function

Specify the Java types of parameters for a Java method.

Format

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

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

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

Syntax Rules

1) A <Java data type> is a Java data type that is mappable or result set mappable, as specified in Subclause 4.5, "Parameter mapping". The <Java data type> names are case sensitive, and shall be fully qualified with their package names, if any.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature J631, "Java signatures", conforming SQL language shall not contain a <Java parameter declaration list> that is not equivalent to the default Java method signature as determined in Subclause 8.6, "Java routine signature determination".

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2 <SQL Java path>

Function

Control the resolution of Java classes across installed JARs.

Format

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

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

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

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

Syntax Rules

None.

Access Rules

None.

General Rules

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

   NOTE 9 — A JAR descriptor's SQL-Java path is set by an invocation of the `SQLJ.ALTER_JAVA_PATH` procedure.

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

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

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

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

      NOTE 10 — This exception can only occur if the implementation-defined action taken for an `SQLJ.ALTER_JAVA_PATH` call that raised an exception results in leaving invalid <jar name>s in the SQL-Java path.

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

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

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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

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

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

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

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

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

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

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

Conformance Rules

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

© ISO/IEC 2008 – All rights reserved

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3 <routine invocation>

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

Function

Invoke an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

1) Insert this 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 to be performed by <routine invocation>, then the Syntax Rules of Subclause 8.6, "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

1) Insert after GR 3)b)i) If *R* is an external Java routine, then let CPV_i be an implementation-defined non-null value of declared type T_i.

2) Insert before GR 4) 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 11 — The subject Java method of an external Java routine is defined in Subclause 8.6, "Java routine signature determination".

3) Replace the lead text of GR 4) If *R* is an external routine that is not an external Java routine, then:

4) Replace the lead text of GR 5)d)ii) If *R* is not a static field method, then:

5) Insert before GR 8)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).

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

NOTE 12 — Initialization of the save area data item occurs in Subclause 8.5, "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.

6) Replace the lead text of GR 8)f)ii)1) If *R* is not an external Java routine and *R* is neither an array-returning external function nor a multiset-returning external function, then *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

7) Insert before GR 8)f)ii)2) If *R* is an external Java routine and *R* is not an array-returning external function or a multiset-returning external function, then *P* is executed in a manner determined as follows and with a list of parameters PD_i whose values are set as follows:

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

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

c) Case:

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

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

NOTE 13 — The "subject static field" of an SQL-invoked method is defined in Subclause 8.6, "Java routine signature determination".

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

NOTE 14 — "effective returns data type" is defined in the Syntax Rules of Subclause 10.4, "<routine invocation>", in [ISO9075-2].

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*.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

      NOTE 15 — If *UIS* is SERIALIZABLE, then, as described in Subclause 9.4, "<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 [J2SE].

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.

      NOTE 16 — If *UIS* is SQLDATA, then, as described in Subclause 9.4, "<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 [J2SE].

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 rules of Subclause 9.2, "Store assignment", in [ISO9075-2], are applied with *SSFV* as *VALUE* and *RESULT* as *TARGET*. The result of the <routine invocation> is the value of *RESULT*. No further General Rules of this Subclause are applied.

ii) Otherwise:

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

   NOTE 17 — 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 18 — The component type of a Java array is defined in [Java].

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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*.

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.

            NOTE 19 — If *UIS* is SERIALIZABLE, then, as described in Subclause 9.4, "<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 [J2SE].

      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.

            NOTE 20 — If *UIS* is SQLDATA, then, as described in Subclause 9.4, "<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 [J2SE].

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

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

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

      NOTE 21 — 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.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

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

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

Case:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
JCLSN PD1 = new
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
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 23 — 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].

8) Insert after GR 8)f)ii)4) 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.

9) Insert before GR 8)g)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*.

10) Replace the 1st sentence of the lead text of GR 8)g)ii) If *R* is not an external Java routine, then for *i* varying from 1 (one) to *EN*, the value of ESP_i is set to the value of PD_i.

11) Insert after GR 8)h)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:

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

NOTE 24 — If *UIS* is SERIALIZABLE, then, as described in Subclause 9.4, "<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 [J2SE].

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.

NOTE 25 — If *UIS* is SQLDATA, then as described in Subclause 9.4, "<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 [J2SE].

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

b) Let *RV* be *SSFV*.

12) Insert after GR 8)h)i)3) If *R* is an external Java routine that is a type-preserving function, then let *ERT* be the effective returns data type of *R*. The returned value of *P*, PD_1, is processed as follows:

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

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

Case:

i) If *UIS* is SERIALIZABLE, then:

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

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

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE 26 — If *UIS* is SERIALIZABLE, then as described in Subclause 9.4, "<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 [J2SE].

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.

NOTE 27 — If *UIS* is SQLDATA, then as described in Subclause 9.4, "<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 [J2SE].

c) Let *RV* be *SSFV*.

13) Insert after GR 8)i)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.

NOTE 28 — If *UIS* is SERIALIZABLE, then as described in Subclause 9.4, "<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 [J2SE].

B) If *UIS* is SQLDATA, then:

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

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

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

NOTE 29 — If *UIS* is SQLDATA, then as described in Subclause 9.4, "<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 [J2SE].

ii) Otherwise, CPV_i is set to EPV_i.

14) Replace GR 10)b) If *R* is not an external Java routine, then let *OPN* be the actual number of returned result sets included in *RSS*.

15) Insert after GR 10)b) 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*.

16) Insert before GR 10)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.

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.

17) Replace GR 10)d) If *R* is not an external Java routine, then for each *i*, 1 (one) ≤ *i* ≤ *RTN*, let FRC_i be the with-return cursor of the *i*-th returned result set RS_i in *RSS*, and let $FRCN_i$ be the <cursor name> that identifies FRC_i.

18) Insert after GR 10)d) 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) ≤ *i* ≤ *RTN*, be the *i*-th cursor in *FRC*, let $FRCN_i$ be the <cursor name> that identifies FRC_i, and let RCS_i be the result set of FRC_i.

19) Replace GR 10)f) If *R* is not an external Java routine, then a completion condition is raised: *warning — result sets returned*.

20) Insert after GR 10)f) 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.

21) Insert before GR 13) If *R* is an external Java routine, then whether the call of *P* returns update counts as defined in JDBC is implementation-defined.

Conformance Rules

1) Insert this CR Without Feature J611, "References", conforming SQL language shall not contain a <reference value expression>.

2) Insert this CR Without Feature J611, "References", conforming SQL language shall not contain a <right arrow>.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.4 <language clause>

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

Function

Specify a programming language.

Format

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

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

Augment Table 15, "Standard programming languages"

Table 1 — Standard programming languages

Language keyword	Relevant standard
JAVA	[Java]

Conformance Rules

No additional Conformance Rules.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

8.5 Execution of array-returning functions

This Subclause modifies Subclause 9.14, "Execution of array-returning functions", in ISO/IEC 9075-2.

Function

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

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 6) Case:

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

   b) Otherwise, let *PN* and *N* be the number of values in the static SQL argument list of *P*.

2) Replace the lead text of GR 8) If *P* is not an external Java routine, and the call type data item has a value of –1 (indicating "open call"), then *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

3) Insert before GR 9) If *P* is an external Java routine, and the call type data item has a value of –1 (indicating "open call"), then *P* is executed with a list of parameters PD_i, 1 (one) ≤ *i* ≤ *EN*, whose values are set as follows:

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

      NOTE 30 — If any Java parameter is declared to be of an array class, then *JPDL* reflects that information.

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

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

      NOTE 31 — The component type of a Java array is defined in [Java].

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

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

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Case:

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

Case:

1) If *UIS* is SERIALIZABLE, then:

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

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

NOTE 32 — If *UIS* is SERIALIZABLE, then, as described in Subclause 9.4, "<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 [J2SE].

2) If *UIS* is SQLDATA, then:

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

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

NOTE 33 — If *UIS* is SQLDATA, then, as described in Subclause 9.4, "<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 [J2SE].

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

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

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

NOTE 34 — PAD_i is a Java object effectively created by execution of the Java expression new JP_i[1].

ii) PD_i is replaced by PAD_i.

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

i) Case:

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

NOTE 35 — PAD_i is a Java object effectively created by execution of the Java expression `new java.lang.String[1]`.

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

ii) PD_i is replaced by PAD_i .

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

h) For the *call type data item*, for *i* equal to *EN*, the value of PD_i is set to the value %ndash;1 (indicating "open call").

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

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

4) Replace GR 9)a) If either *P* is not an external Java routine and the value of the exception data item is '00000' (corresponding to the completion condition *successful completion*), or *P* is not an external Java routine and the first 2 characters of the exception data item are '01' (corresponding to the completion condition *warning* with any subcondition), then set the call type data item to 0 (zero) (indicating *fetch call*).

5) Insert before GR 9)b) If *P* is an external Java routine and the prior invocation of *P* did not terminate with an unhandled Java exception, then set the call type data item to 0 (zero) (indicating *fetch call*).

6) Replace the lead text of GR 9)b) If *P* is not an external Java routine and the value of the exception data item is '02000' (corresponding to the completion condition *no data*):

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

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

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

8) Replace the lead text of GR 10)a) If *P* is not an external Java routine, then *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

9) Insert before GR 10)b) If *P* is an external Java routine, then *P* is executed with a list of *EN* parameters PD_i and whose values are set as follows:

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

Case:

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

Case:

1) If *UIS* is SERIALIZABLE, then:

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

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

NOTE 36 — If UIS is SERIALIZABLE, then, as described in Subclause 9.4, "<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 [J2SE].

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2) If *UIS* is SQLDATA, then:

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

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

NOTE 37 — If *UIS* is SQLDATA, then, as described in Subclause 9.4, "<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 [J2SE].

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

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

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

NOTE 38 — PAD_i is a Java object effectively created by execution of the Java expression new JP_i [1].

ii) PD_i is replaced by PAD_i .

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

i) Case:

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

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

ii) PD_i is replaced by PAD_i.

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

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

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

10) Replace the lead text of GR 10)b)i) If *P* is not an external Java routine and either the exception data item is '00000' (corresponding to completion condition *successful completion*) or the first 2 characters are '01' (corresponding to completion condition *warning* with any subcondition), or P is an external Java routine and the prior invocation of *P* did not terminate with an unhandled Java exception, then:

11) Replace GR 10)b)i)3)A) If *P* is not an external Java routine and each PD_i for *i* ranging from (*PN*+*FRN*)+*N*+1 through (*PN*+*FRN*)+*N*+*FRN* (that is, the SQL indicator arguments corresponding to the result data items), is negative, then let the *E*-th element of *AR* be the null value.

12) Insert after GR 10)b)i)3)A) If *P* is an external Java routine and each JP_i for *i* ranging from *PN*+1 to *PN*+*FRN* that is a Java class has an associated value of the first element, ([0]), of PD_i that is a Java null, then let the *E*-th element of *AR* be the null value.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13) Replace GR 10)b)i)3)B)I) If *P* is not an external Java routine and *FRN* is 1 (one), then let the *E*-th element of *AR* be the value of the result data item.

14) Insert after GR 10)b)i)3)B)I) If *P* is an external Java routine, then for the result data items, for i ranging from *PN*+1 through *PN*+*FRN*:

a) Case:

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

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

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

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

4) Case:

A) If *UIS* is SERIALIZABLE, then:

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

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

NOTE 39 — If *UIS* is SERIALIZABLE, then, as described in Subclause 9.4, "<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 [J2SE].

B) If *UIS* is SQLDATA, then:

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

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

NOTE 40 — If *UIS* is SQLDATA, then as described in Subclause 9.4, "<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 [J2SE].

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

b) Case:

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

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

`ROW (` SV_1`, ...,` SV_{FRN} `)`

15) Replace the lead text of GR 10)b)ii) If *P* is not an external Java routine and the exception data item is '02000' (corresponding to completion condition *no data*), or if *P* is an external Java routine and the prior

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

16) Replace the lead text of GR 11) If *P* is not an external Java routine, and the call type data item has a value of 1 (one) (indicating *close call*), then *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

17) Insert after GR 11) If *P* is an external Java routine and the call type data item has a value of 1 (one) (indicating close call), then *P* is executed with a list of *EN* parameters PD_i and whose values are set as follows:

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

Case:

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

Case:

1) If *UIS* is SERIALIZABLE, then:

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

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

NOTE 41 — If *UIS* is SERIALIZABLE, then, as described in Subclause 9.4, "<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 [J2SE].

2) If *UIS* is SQLDATA, then:

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

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

NOTE 42 — If *UIS* is SQLDATA, then, as described in Subclause 9.4, "<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 [J2SE].

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

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

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

NOTE 43 — PAD_i is a Java object effectively created by execution of the Java expression `new` JP_i `[1]`.

ii) PD_i is replaced by PAD_i .

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

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

i) Case:

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

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

ii) PD_i is replaced by PAD_i .

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

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

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

Conformance Rules

No additional Conformance Rules.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.6 Java routine signature determination

Function

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

Syntax Rules

1) Let *SE*, *i*, and *SR* respectively be *ELEMENT* (the syntactic element), *INDEX* (the method specification index), and *SUBJECT* (the subject routine (if any)) specified in an application of this Subclause.

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

   Case:

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

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

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

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

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

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

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

         1) Let *RDPL* be a result data area parameter list that specifies a comma-seperated list of <SQL parameter declaration>s that have <parameter mode> OUT; their <parameter type>s are defined to be those of the effective SQL parameter list entries *PN*+1 through *PN*+*FRN* as defined in Subclause 11.51, "<SQL-invoked routine>".

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

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

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

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

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

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

iii) For the purposes of parameter mapping as defined in Subclause 4.5, "Parameter mapping", the remaining rules in this Subclause are performed as if the descriptor for the user-defined type defined by *UDTD* was already available in the SQL-session. That descriptor describes the type as having the name *UDTN*, being an external Java data type, and having the <jar and class name> specified in *UDTD*.

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

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

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

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

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

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

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

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

( )

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

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

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

3) If MS_i specifies an array-returning external function or a multiset-returning external function then:

A) Let *RDPL* be a result data area parameter list that specifies a comma-seperated list of <SQL parameter declaration>s that have <parameter mode> OUT; their <parameter type>s are defined to be those of the effective SQL parameter list entries *PN*+1 through *PN*+*FRN* as defined in Subclause 9.8, "<SQL-invoked routine>".

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

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

 © ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

c) Otherwise, descriptors are available.

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

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

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

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

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

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

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

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

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

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

( )

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

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

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

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

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

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

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

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

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

A) Let *RDPL* be a result data area parameter list that specifies a comma-seperated list of <SQL parameter declaration>s that have <parameter mode> OUT; their <parameter type>s are defined to be those of the effective SQL parameter list entries *PN*+1 through *PN*+*FRN* as defined in Subclause 9.8, "<SQL-invoked routine>".

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

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

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

3) Case:

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

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

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

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

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

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

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

v) *SPDL* shall specify either:

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

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

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

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

NOTE 45 — "visible" is defined in Subclause 4.5, "Parameter mapping".

b) Otherwise:

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

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

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

Case:

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

```
JCLSN ,
```

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

```
( JCLSN )
```

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

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

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

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

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

Case:

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

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

NOTE 46 — “simply mappable”, “object mappable”, and “array mappable” are defined in Subclause 4.5, “Parameter mapping”.

3) Case:

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

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

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

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

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

SP; if *SP* is an SQL array, then let *JT* be the corresponding Java data type of the <parameter type> of *SP*, and let the corresponding Java parameter data type of *SP* be an array of *JT*, that is, be `JT[]`.

NOTE 47 — The "corresponding Java parameter data type" of *SP* is defined in Subclause 4.5, "Parameter mapping".

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

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

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

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

Case:

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

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

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

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

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

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

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

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

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

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

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

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

NOTE 49 — "simply mappable" and "object mappable" are defined in Subclause 4.5, "Parameter mapping".

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

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

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

I) Case:

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

Case:

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

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

( )

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

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

NOTE 51 — "visible" is defined in Subclause 4.5, "Parameter mapping".

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

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

4) The subject Java method is determined as follows:

Case:

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

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

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

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

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

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

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

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

Case:

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

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

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

iii) If *JM* is an array-returning external function or a multiset-returning external function, then *JRT* shall be `void`.

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

NOTE 52 — “simply mappable” and “object mappable” are defined in Subclause 4.5, “Parameter mapping”.

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

Access Rules

None.

General Rules

None.

Conformance Rules

None.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9 Schema definition and manipulation

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

9.1 <drop schema statement>

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

Function

Destroy a schema.

Format

No additional Format items.

Syntax Rules

1) Add JARs to the list of objects in SR 4)

Access Rules

No additional Access Rules.

General Rules

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

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

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

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

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

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

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

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

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance Rules

No additional Conformance Rules.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.2 <table definition>

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

Function

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

Format

```
No additional Format items.
```

Syntax Rules

1) Insert after SR 8)e) *ST* shall not be an external Java data type whose descriptor specifies an <interface specification> of SERIALIZABLE.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3 <view definition>

This Subclause modifies Subclause 11.23, "<view definition>", in ISO/IEC 9075-2.

Function

Define a viewed table.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 20)c) *ST* shall not be an external Java data type whose descriptor specifies an <interface specification> of SERIALIZABLE.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.4 <user-defined type definition>

This Subclause modifies Subclause 11.42, "<user-defined type definition>", in ISO/IEC 9075-2.

Function

Define a user-defined type.

Format

```
<user-defined type body> ::=
  <schema-resolved user-defined type name> [ <subtype clause> ]
      [ <external Java type clause> ]
      [ AS <representation> ]
      [ <user-defined type option list> ] [ <method specification list> ]

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

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

<interface specification> ::=
    SQLDATA
  | SERIALIZABLE

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

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

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

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

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

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

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

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

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Syntax Rules

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

2) Replace SR 8)j)ii) The <supertype name> immediately contained in the <subtype clause> shall identify the descriptor of some structured type *SST*. *UDT* is a direct subtype of *SST*, and *SST* is a direct supertype of *UDT*. If *UDT* is an external Java data type, then *SST* shall be an external Java data type, and the subject Java class of *UDT* shall be a direct subclass of the subject Java class of *SST*. If *UDT* is not an external Java data type, then *SST* shall not be an external Java data type.

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

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

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

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

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

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

   e) If SQLDATA is specified, then the subject Java class shall implement the Java interface `java.sql.SQLData` as defined in [JDBC] and [J2SE]. The method `java.sql.SQLData.writeSQL()` is effectively used to convert a Java object to an SQL representation, and the method`java.sql.SQLData.readSQL()` is effectively used to convert an SQL representation to a Java object.

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

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

   h) SELF AS LOCATOR shall not be specified.

   i) <locator indication> shall not be specified.

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

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

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

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

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

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

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

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

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

   a) MS_i specifies a *static field method*.

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

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

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

11) Insert after SR 9)b)xiv)1) If *UDT* is an external Java data type, then let *VMP* be the value of the <character string literal> immediately contained in <external Java method clause>; *VMP* shall conform to the Format and Syntax Rules of <Java method and parameter declarations>.

12) Replace SR 9)b)xiv)2) If *UDT* is not an external Java data type and <language clause> is not specified, then LANGUAGE SQL is implicit.

13) Replace SR 9)b)xiv)6)B)I) If <parameter style> is not specified and *UDT* is not an external Java data type, then PARAMETER STYLE SQL is implicit.

14) Insert after SR 9)b)xv) If *UDT* is an external Java data type and validation of the <Java parameter declaration list> has been implementation-defined to be performed by <user-defined type definition>, then the Syntax Rules of Subclause 8.6, "Java routine signature determination", are applied with <user-defined type definition> as *ELEMENT*, *i* as *INDEX*, and no subject routine as *SUBJECT*.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 1)g)vii) The explicit or implicit <parameter style> if the <language name> is SQL or JAVA.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance Rules

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

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

3) Insert this CR Without Feature J641, “Static fields”, conforming SQL language shall not contain a <static field method spec>.

4) Insert this CR Without Feature J541, “SERIALIZABLE”, conforming SQL language shall not contain an <interface specification> that contains SERIALIZABLE.

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

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

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5 <attribute definition>

This Subclause modifies Subclause 11.43, "<attribute definition>", in ISO/IEC 9075-2.

Function

Define an attribute of a structured type.

Format

```
<attribute definition> ::=
  <attribute name> <data type>
      [ <reference scope check> ] [ <attribute default> ]
      [ <collate clause> ] [ <external Java attribute clause> ]

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

Syntax Rules

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

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

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

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

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

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 3)e) If the <attribute definition> contains an <external Java attribute clause>, then the corresponding Java field name of the <attribute name>.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2) Replace GR 4) An SQL-invoked method *OF* is created whose signature and result data type are as given in the descriptor of the original method specification of the observer function of *A*. Let *V* be a value in *UDT*.

Case

a) If *V* is the SQL null value, then the invocation `V.AN()` of *OF* returns the result of:

```
CAST (NULL AS DT)
```

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

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

Case:

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

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

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

3) Case:

A) If *UIS* is SERIALIZABLE, then:

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

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

NOTE 55 — If *UIS* is SERIALIZABLE, then, as described in Subclause 9.4, "<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 [J2SE].

B) If *UIS* is SQLDATA, then:

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

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

NOTE 56 — If *UIS* is SQLDATA, then, as described in Subclause 9.4, "<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 [J2SE].

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

4) `V.AN( )` returns the value of *SV*.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3) Replace GR 5) An SQL-invoked method *MF* is created whose signature and result data type are as given in the descriptor of the original method specification of the mutator function of *A*. Let *V* be a value in *UDT* and let *AV* be a value in *DT*.

Case:

a) If *V* is the SQL null value, then the invocation `V.AN(AV)` of *MF* raises an exception condition: *data exception — null instance used in mutator function*.

b) If *UDT* is not an external Java data type whose descriptor's <interface specification> specifies SERIALIZABLE, then the invocation `V.AN(AV)` returns *V2* such that `V2.AN() = AV` and for every other observer function *ANX* of *UDT*, `V2.ANX() = V.ANX()`.

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

i) Case:

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

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

B) Case:

I) If *UIS* is SERIALIZABLE, then:

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

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

NOTE 57 — If *UIS* is SERIALIZABLE, then, as described in Subclause 9.4, "<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 [J2SE].

II) If *UIS* is SQLDATA, then:

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

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

NOTE 58 — If *UIS* is SQLDATA, then, as described in Subclause 9.4, "<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 [J2SE].

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

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

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

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

Conformance Rules

No additional Conformance Rules.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.6 <alter type statement>

This Subclause modifies Subclause 11.44, "<alter type statement>", in ISO/IEC 9075-2.

Function

Change the definition of a user-defined type.

Format

```
No additional Format items.
```

Syntax Rules

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

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.7 <drop data type statement>

This Subclause modifies Subclause 11.50, "<drop data type statement>", in ISO/IEC 9075-2.

Function

Destroy a user-defined type.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

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

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

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.8 <SQL-invoked routine>

This Subclause modifies Subclause 11.51, "<SQL-invoked routine>", in ISO/IEC 9075-2.

Function

Define an SQL-invoked routine.

Format

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

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

Syntax Rules

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

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

3) Insert after SR 3) If <SQL-invoked routine> specifies LANGUAGE JAVA, then <transform group specification> shall not be specified.

4) Insert after SR 3) The maximum value of <maximum returned result sets> is implementation-defined.

5) Replace SR 5)b)i) Let *UDTN* be the <schema-resolved user-defined type name> immediately contained in <method specification designator>. Let *UDT* be the user-defined type identified by *UDTN*. *UDT* shall not be an external Java type.

6) Replace SR 6)a) <routine characteristics> shall contain at most one <language clause>, at most one <parameter style clause>, at most one <specific name>, at most one <deterministic characteristic>, at most one <SQL-data access indication>, at most one <null-call clause>, and at most one <returned result sets characteristic>. If LANGUAGE JAVA is specified, then <parameter style clause> shall specify <parameter style> JAVA.

7) Replace SR 6)i) An <SQL-invoked routine> that specifies or implies LANGUAGE SQL is called an *SQL routine*; an <SQL-invoked routine> that does not specify LANGUAGE SQL is called an *external routine*. An external routine that specifies LANGUAGE JAVA is called an *external Java routine*.

8) Insert after SR 6)i) If *R* is an external Java routine, then the <external routine name> immediately contained in <external body reference> shall specify a <character string literal>. Let *V* be the value of that <character string literal>. *V* shall conform to the Format and Syntax Rules of an <external Java reference string>.

NOTE 59 — *R* is defined by [ISO9075-2] to be the SQL-invoked routine specified by <SQL-invoked routine>.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9) Insert after SR 6)i) If *R* is an external Java routine, then the <Java method name> is the name of one or more Java methods in the class specified by <Java class name> in the JAR specified by <jar name>. The combination of <Java class name> and <Java method name> represent a fully qualified Java class name and method name. The method name can reference a method of the class, or a method of a superclass of the class.

10) Replace SR 6)x)ii) If *R* is an array-returning external function or a multiset-returning external function that is not an external Java routine, then PARAMETER STYLE SQL shall be either specified or implied.

11) Replace the lead text of SR 6)x)iii) If *R* is not an external Java routine, then

Case:

12) Insert before SR 20)e) If PARAMETER STYLE JAVA is specified, then:

a) Case:

i) If *R* is an array-returning external function or a multiset-returning external function and the returned array's element type or returned multiset's element type is a row type, then let *FRN* be the degree of the element type.

ii) Otherwise, let *FRN* be 1 (one).

b) If *R* is an array-returning external function or a multiset-returning external function, then let *AREF* be 2. Otherwise, let *AREF* be 0 (zero).

c) If *R* is an SQL-invoked function, then let the *effective SQL parameter list* be a list of *PN*+*FRN*+*AREF* SQL parameters, as follows:

i) For *i* ranging from 1 (one) to *PN*, the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration>.

ii) Case:

1) If *FRN* is 1 (one), then effective SQL parameter list entry *PN*+*FRN* has <parameter mode> OUT; its <parameter type> *PT* is defined as follows:

A) If <result cast> is specified, then let *RT* be <result cast from type>; otherwise, let *RT* be <returns data type>.

B) If *R* is an array-returning external function or a multiset-returning external function, then let *PT* be the element type of *RT*.

C) If *R* is neither an array-returning external function nor a multiset-returning external function, then *PT* is *RT*.

2) Otherwise, for *i* ranging from *PN*+1 to *PN*+*FRN*, the *i*-th effective SQL parameter list entry is defined as follows:

A) Its <parameter mode> is OUT.

B) Let $RFT_{i\text{-}PN}$ be the data type of the (*i*–*PN*)-th field of the element type of the <returns data type>. The <parameter type> PT_i of the *i*-th effective SQL parameter list entry is $RFT_{i\text{-}PN}$.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

iii) If *R* is an array-returning external function or a multiset-returning external function, then:

1) Effective SQL parameter type list entry (*PN*+*FRN*)+1 is an SQL parameter whose <data type> is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> INOUT.

2) Effective SQL parameter type list entry (*PN*+*FRN*)+2 is an SQL parameter whose <data type> is an exact numeric type with scale 0 (zero) and with <parameter mode> IN.

d) If *R* is an SQL-invoked procedure, then let the effective SQL parameter list be a list of *PN* SQL parameters. For *i* ranging from 1 (one) to *PN*, the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration>.

13) Replace SR 20)g) If <language clause> does not specify JAVA, then every <data type> in an effective SQL parameter list entry shall specify a data type listed in the SQL data type column for which the corresponding row in the host data type column is not 'None'.

14) Insert before SR 21)

NOTE 60 — The rules for parameter type correspondence when LANGUAGE JAVA is specified are given in Subclause 4.5, "Parameter mapping".

15) Insert before SR 21) If *R* is an external Java routine, then it is implementation-defined whether validation of the explicit or implicit <Java parameter declaration list> is performed by <SQL-invoked routine> or when its SQL-invoked routine is invoked.

16) Insert before SR 21) If *R* is an external Java routine, and validation of the <Java parameter declaration list> has been implementation-defined to be performed by <SQL-invoked routine>, then the Syntax Rules of Subclause 8.6, "Java routine signature determination", are applied with <SQL-invoked routine> as *ELEMENT*, 0 (zero) as *INDEX*, and no subject routine as *SUBJECT*.

Access Rules

1) Insert after AR 1) If *R* is an external Java routine, then the applicable privileges for *A* shall include USAGE privilege on the JAR referenced in the <external Java reference string>.

NOTE 61 — The references to *R* and *A* are defined in the Syntax Rules of Subclause 11.51, "<SQL-invoked routine>", in [ISO9075-2].

General Rules

1) Replace GR 3)l)ii) The routine descriptor includes an indication of whether the parameter passing style is PARAMETER STYLE JAVA, PARAMETER STYLE SQL, or PARAMETER STYLE GENERAL.

2) Replace the lead text of GR 6)a)i) If *R* is not an external Java routine and the <SQL data access indication> in the descriptor of *R* is MODIFIES SQL DATA, READS SQL DATA, or CONTAINS SQL, then:

Conformance Rules

1) Insert this CR Without Feature J511, "Commands", conforming SQL language shall not contain an <SQL-invoked routine> that contains a <language name> that contains JAVA and that is not contained in a <descriptor file>.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2) Insert this CR Without Feature J581, “Output parameters”, conforming SQL language shall not contain an <SQL-invoked routine> that contains a <language name> that contains JAVA and that contains a <parameter mode> that contains either OUT or INOUT.

3) Insert this CR Without Feature J521, “JDBC data types”, conforming SQL language shall not contain a <Java data type> that is not the corresponding Java data type of some SQL data type.

4) Insert this CR Without Feature J621, “external Java routines”, conforming SQL language shall not contain an <SQL-invoked routine> that contains a <language name> that contains JAVA.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.9 <alter routine statement>

This Subclause modifies Subclause 11.52, "<alter routine statement>", in ISO/IEC 9075-2.

Function

Alter a characteristic of an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 1) *SR* shall not be an external Java routine.

NOTE 62 — *SR* is defined to be the SQL-invoked routine identified by the <alter routine statement>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

9.10 <drop routine statement>

This Subclause modifies Subclause 11.53, "<drop routine statement>", in ISO/IEC 9075-2.

Function

Destroy an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

1) Insert this SR If *SR* is an external Java routine and <drop routine statement> is contained in a <descriptor file>, then <drop routine statement> shall specify a <routine type> of PROCEDURE or of FUNCTION.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature J511, "Commands", conforming SQL language shall not contain a <drop routine statement> that contains a <specific routine designator> that identifies an external Java routine and that is not contained in a <descriptor file>.

2) Insert this CR Without Feature J621, "external Java routines", conforming SQL language shall not contain a <drop routine statement> that contains a <specific routine designator> that identifies an external Java routine.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.11 <user-defined ordering definition>

This Subclause modifies Subclause 11.56, "<user-defined ordering definition>", in ISO/IEC 9075-2.

Function

Define a user-defined ordering for a user-defined type.

Format

```
<ordering category> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <comparable category>

<comparable category> ::=
  RELATIVE WITH COMPARABLE INTERFACE
```

Syntax Rules

1) Replace SR 4) If <comparable category>, <relative category>, or <state category> is specified, then *UDT* shall be a maximal supertype.

2) Insert before SR 6) If <comparable category> is specified, then *UDT* shall be an external Java data type. Let *JC* be the subject Java class of that external Java data type. *JC* shall implement the Java interface `java.lang.Comparable`.

3) Replace the lead text of SR 6)b) If <comparable category> is not specified, then:

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 3)c) If <comparable category> is specified, then the ordering category in the user-defined type descriptor of *UDT* is set to COMPARABLE.

Conformance Rules

1) Insert this CR Without Feature J622, "external Java types", conforming SQL language shall not contain a <user-defined ordering definition> that contains a <schema-resolved user-defined type name> that identifies an external Java type.

2) Insert this CR Without Feature J511, "Commands", conforming SQL language shall not contain a <user-defined ordering definition> that contains a <schema-resolved user-defined type name> that identifies an external Java type and that is not contained in a <descriptor file>.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.12 <drop user-defined ordering statement>

This Subclause modifies Subclause 11.57, "<drop user-defined ordering statement>", in ISO/IEC 9075-2.

Function

Destroy a user-defined ordering method.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature J622, "external Java types", conforming SQL language shall not contain a <drop user-defined ordering statement> that contains a <schema-resolved user-defined type name> that identifies an external Java type.

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

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10 Access control

This Clause modifies Clause 12, "Access control", in ISO/IEC 9075-2.

10.1 <grant privilege statement>

This Subclause modifies Subclause 12.2, "<grant privilege statement>", in ISO/IEC 9075-2.

Function

Define privileges.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature J511, "Commands", conforming SQL language shall not contain a <grant privilege statement> that contains an <object name> that immediately contains a <jar name> and that is not contained in a <descriptor file>.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.2 <privileges>

This Subclause modifies Subclause 12.3, "<privileges>", in ISO/IEC 9075-2.

Function

Specify privileges.

Format

```
<object name> ::=
    !! All alternatives from ISO/IEC 9075-2
  | JAR <jar name>
```

Syntax Rules

1) Replace SR 3) If <object name> specifies a <domain name>, <collation name>, <character set name>, <transliteration name>, <schema-resolved user-defined type name>, <sequence generator name>, or <jar name>, then <privileges> shall specify USAGE. Otherwise, USAGE shall not be specified.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature J561, "JAR privileges", conforming SQL language shall not contain an <object name> that immediately contains a <jar name>.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.3 <revoke statement>

This Subclause modifies Subclause 12.7, "<revoke statement>", in ISO/IEC 9075-2.

Function

Destroy privileges and role authorizations.

Format

```
No additional Format items.
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 5)b)i)3)D) *P* and *D* are both usage privilege descriptors. The action and the identified domain, character set, collation, transliteration, user-defined type, sequence generator, or JAR of *P* are the same as the action and the identified domain, character set, collation, transliteration, user-defined type, sequence generator, or JAR of *D*, respectively.

2) Insert after GR 24)b) *DT* is an external Java data type and the revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE on the JAR whose <jar name> is contained in the <jar and class name> of the descriptor of *DT*.

3) Insert after GR 28) Let *JR* be any JAR descriptor included in *S1*. *JR* is said to be *impacted* if the revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on a JAR whose name is contained in a <resolution jar> contained in the SQL-Java path of *JR*.

4) Insert after GR 29)r) If *RD* is an external Java routine, USAGE on the JAR whose <jar name> is contained in <external Java reference string> contained in the <external routine name> of the descriptor of *RD*.

5) Insert after GR 31) If RESTRICT is specified, and there exists an impacted JAR, then an exception condition is raised: *dependent privilege descriptors still exist*.

6) Insert after GR 48) If the object identified by <object name> of the <revoke statement> specifies <jar name>, let *J* be the JAR identified by that <jar name>. For every impacted JAR descriptor *JR* and for each <path element> *PE* contained in the SQL-Java path of *JR* whose immediately contained <resolution jar> is *J*, the SQL-Java path of the JAR descriptor *JR* is modified such that it does not contain *PE*.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance Rules

1) Insert this CR Without Feature J511, "Commands", conforming SQL language shall not contain a <revoke statement> that an <object name> that immediately contains a <jar name> and that is not contained in a <descriptor file>.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11 Built-in procedures

11.1 SQLJ.INSTALL_JAR procedure

Function

Install a set of Java classes into the current SQL catalog and schema.

Signature

```
SQLJ.INSTALL_JAR (
    url      IN    CHARACTER VARYING(L),
    jar      IN    CHARACTER VARYING(L),
    deploy   IN    INTEGER )
```

Where *L* is an implementation-defined integer value.

Access Rules

1) The privileges required to invoke the `SQLJ.INSTALL_JAR` procedure are implementation-defined.

General Rules

1) The `SQLJ.INSTALL_JAR` procedure is subject to implementation-defined rules for executing SQL-schema statements within SQL-transactions. If an invocation of `SQLJ.INSTALL_JAR` raises an exception condition, then the effect on the install actions is implementation-defined.

2) The values of the `url` parameter that are valid are implementation-defined, and may include URLs whose format is implementation-defined. If the value of the `url` parameter does not conform to implementation-defined restrictions and does not identify a valid JAR, then an exception condition is raised: *Java DDL — invalid URL*.

3) Let *J* be the value of the `jar` parameter. Let *TJ* be the value of

   ```
   TRIM ( BOTH ' ' FROM J )
   ```

   If *TJ* does not conform to the Format and Syntax Rules of <jar name>, then an exception condition is raised: *Java DDL — invalid JAR name*.

4) Let *JN* be the explicitly or implicitly qualified <jar id> specified in the <jar name> specified by *TJ*.

5) If there is an installed JAR whose name is *JN*, then an exception condition is raised: *Java DDL — invalid JAR name*.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6) The JAR is installed and associated with the name *JN*. All contents of the JAR are installed, including both visible and non-visible Java classes, and other items contained in the JAR. This JAR becomes the *associated JAR* of each new class. The non-visible Java classes and other items can be referenced by other Java methods.

7) A JAR descriptor is created that describes the JAR being installed. The JAR descriptor includes the name of the JAR, and an empty SQL-Java path.

8) A privilege descriptor is created that defines the USAGE privilege on the JAR identified by the `jar` parameter to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the `jar` parameter. The grantor for the privilege descriptor is set to the special grantor value "_SYSTEM". The privilege is grantable.

9) If the value of the `deploy` parameter is not zero, and if the JAR contains one or more deployment descriptor files, then the install actions implied by those instances are performed in the order in which the deployment descriptor files appear in the manifest.

NOTE 63 — Deployment descriptor files and their install actions are specified in Subclause 4.11.1, "Deployment descriptor files".

Conformance Rules

1) Without Feature J531, "Deployment", conforming SQL language shall not contain invocations of the SQLJ.INSTALL_JAR procedure that provide non-zero values of the `deploy` parameter.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.2 SQLJ.REPLACE_JAR procedure

Function

Replace an installed JAR.

Signature

```
SQLJ.REPLACE_JAR (
    url      IN    CHARACTER VARYING (L),
    jar      IN    CHARACTER VARYING (L) )
```

Where: *L* is an implementation-defined integer value.

Access Rules

1) The privileges required to invoke the SQLJ.REPLACE_JAR procedure are implementation-defined.

2) The current user shall be the owner of the JAR specified by the value of the `jar` parameter.

General Rules

1) The SQLJ.REPLACE_JAR procedure is subject to implementation-defined rules for executing SQL-schema statements within SQL-transactions.

2) The values of the `url` parameter that are valid are implementation-defined, and may include URLs whose format is implementation-defined. If the value of `url` identifies a valid JAR, then refer to the classes in that JAR as the *new classes*. If the value of the `url` parameter does not identify a valid JAR, then an exception condition is raised: *Java DDL — invalid URL*.

3) Let *J* be the value of the `jar` parameter. Let *TJ* be the value of

   ```
   TRIM ( BOTH ' ' FROM J )
   ```

   If *TJ* does not conform to the format of <jar name>, then an exception condition is raised: *Java DDL — invalid JAR name*.

4) Let *JN* be the explicitly or implicitly qualified <jar id> specified in the <jar name> specified by *TJ*.

5) If there is an installed JAR with <jar name> *JN*, then refer to that JAR as the *old JAR*. Refer to the classes in the old JAR as the *old classes*. If there is not an installed JAR with <jar name> *JN*, then an exception condition is raised: *Java DDL — attempt to replace uninstalled JAR*. Equivalence of JAR names is determined by the rules for equivalence of identifiers as specified in Subclause 5.2, "<token> and <separator>", in [ISO9075-2].

6) Let the *matching old classes* be the old classes whose fully qualified class names are the names of new classes and let the *matching new classes* be the new classes whose fully qualified class names are the names of old classes. Let the *unmatched old classes* be the old classes that are not matching old classes and let the *unmatched new classes* be the new classes that are not matching new classes.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7) Let the *dependent SQL routines* of a JAR be the routines whose descriptor's <external routine name> specifies an <external Java reference string> whose immediately contained <jar name> is equivalent to the JAR name of that JAR.

8) If any dependent SQL routine of the old JAR references a method in an unmatched old class, then an exception condition is raised: *Java DDL — invalid class deletion*.

NOTE 64 — This rule prohibits deleting classes that are referenced by external Java routines. This prohibition does not, however, prevent deletion of classes that are referenced only indirectly by other Java classes.

9) For each dependent SQL routine of the old JAR that references a method in a matching old class, let *CS* be the <SQL-invoked routine> that created the SQL routine. If *CS* is not a valid <SQL-invoked routine> for the corresponding new routine, then an exception condition is raised: *Java DDL — invalid replacement*.

10) Let the *dependent SQL types* of a JAR file be the external Java data types that have as their subject Java class a Java class contained in that JAR.

NOTE 65 — "subject Java class" is defined in Subclause 9.4, "<user-defined type definition>".

11) If there are any dependent SQL types of the specified JAR file that are unmatched old classes, then an exception condition is raised: *Java DDL — invalid class deletion*.

NOTE 66 — This rule prohibits deleting classes that are referenced by external Java data types. This prohibition does not, however, prevent deletion of classes that are referenced only indirectly by other Java classes.

12) For each dependent SQL type, let *CT* be the <user-defined type definition> that created the SQL type. If *CT* is not a valid <user-defined type definition> for the corresponding new class, then an exception condition is raised: *Java DDL — invalid replacement*.

13) The old JAR and all visible and non-visible old classes contained in it are deleted.

14) The new JAR and all visible and non-visible new classes are installed and associated with the specified <jar name>. That JAR becomes the *associated JAR* of each new class. All contents of the new JAR are installed, including both visible and non-visible Java classes, and other items contained in the JAR. The non-visible Java classes and other items can be referenced by other Java methods.

15) The effect of SQLJ.REPLACE_JAR on currently executing SQL statements that use an SQL routine or structured type whose implementation has been replaced is implementation-dependent.

Conformance Rules

None.

 © ISO/IEC 2008 – All rights reserved

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.3 SQLJ.REMOVE_JAR procedure

Function

Remove an installed JAR and its classes.

Signature

```
SQLJ.REMOVE_JAR (
    jar      IN    CHARACTER VARYING (L),
    undeploy IN    INTEGER )
```

Where: *L* is an implementation-defined integer value.

Access Rules

1) The privileges required to invoke the `SQLJ.REMOVE_JAR` procedure are implementation-defined.

2) The current user shall be the owner of the JAR specified by the value of the `jar` parameter.

General Rules

1) The `SQLJ.REMOVE_JAR` procedure is subject to implementation-defined rules for executing SQL-schema statements within SQL-transactions. If an invocation of `SQLJ.REMOVE_JAR` raises an exception condition, then the effect on the remove actions is implementation-defined.

2) Let *J* be the value of the `jar` parameter. Let *TJ* be the value of

   ```
   TRIM ( BOTH ' ' FROM J )
   ```

   If *TJ* does not conform to the format of <jar name>, then an exception condition is raised: *Java DDL — invalid JAR name*.

3) Let *JN* be the explicitly or implicitly qualified <jar id> specified in the <jar name> specified by *TJ*.

4) If there is not an installed JAR with <jar name> *JN*, then an exception condition is raised: *Java DDL — attempt to remove uninstalled JAR*. Equivalence of <jar name>s is determined by the rules for equivalence of identifiers as specified in Subclause 5.2, "<token> and <separator>", in [ISO9075-2].

5) Let *JR* be the JAR identified by *JN*.

6) If the value of the `undeploy` parameter is not 0 (zero), and if *JR* contains one or more deployment descriptor files, then the remove actions implied by those instances are performed in the reverse of the order in which the deployment descriptor files appear in the manifest.

   NOTE 67 — Deployment descriptor files and their remove actions are specified in Subclause 4.11.1, "Deployment descriptor files".

   NOTE 68 — These actions are performed prior to examining the condition specified in the following step.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7) Let the *dependent SQL routines* of a JAR be the routines whose descriptor's <external routine name> specifies an <external Java reference string> whose immediately contained <jar name> is equivalent to the name of that JAR.

8) If there are any dependent SQL routines of *JR*, then an exception condition is raised: *Java DDL — invalid class deletion*.

NOTE 69 — This rule prohibits deleting classes that are referenced by external Java routines. This prohibition does not, however, prevent deletion of classes that are referenced only indirectly by other Java classes.

9) Let the *dependent SQL types* of a JAR be the external Java data types that have as their subject Java class a Java class contained in that JAR.

NOTE 70 — "Subject Java class" is defined in Subclause 9.4, "<user-defined type definition>".

10) If there are any dependent SQL types of *JR*, then an exception condition is raised: *Java DDL — invalid class deletion*.

NOTE 71 — This rule prohibits deleting classes that are referenced by external Java data types. This prohibition does not, however, prevent deletion of classes that are referenced only indirectly by other Java classes.

11) Let the *dependent JARs* of a JAR be the JARs whose descriptors specify an SQL-Java path that immediately contains one or more <path element>s whose contained <jar name> is equivalent to the name of that JAR.

12) If there are any dependent JARs of *JR*, then an exception condition is raised: *Java DDL — invalid JAR removal*.

13) *JR* and all visible and non-visible classes contained in it are deleted.

14) The USAGE privilege on *JR* is revoked from all users that have it.

15) The descriptor of *JR* is destroyed.

16) The effect of `SQLJ.REMOVE_JAR` on currently executing SQL statements that use an SQL routine or structured type whose implementation has been removed is implementation-dependent.

Conformance Rules

1) Without Feature J531, "Deployment", conforming SQL language shall not contain invocations of the SQLJ.REMOVE_JAR procedure that provide non-zero values of the `undeploy` parameter.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11.4 SQLJ.ALTER_JAVA_PATH procedure

Function

Alter the SQL-Java path of a JAR.

Signature

```
SQLJ.ALTER_JAVA_PATH (
    jar      IN    CHARACTER VARYING (L),
    path     IN    CHARACTER VARYING (L) )
```

Where: *L* is an implementation-defined integer value.

Access Rules

1) The privileges required to invoke the SQLJ.ALTER_JAVA_PATH procedure are implementation-defined.
2) The current user shall be the owner of the JAR specified by the value of the `jar` parameter.
3) The current user shall have the USAGE privilege on each JAR referenced in the `path` parameter.

General Rules

1) The SQLJ.ALTER_JAVA_PATH procedure is subject to implementation-defined rules for executing SQL-schema statements within SQL-transactions.
2) Let *J* be the value of the `jar` parameter. Let *TJ* be the value of

   ```
   TRIM ( BOTH ' ' FROM J )
   ```

   If *TJ* does not conform to the format of <jar name>, then an exception condition is raised: *Java DDL — invalid JAR name*.
3) Let *JN* be the explicitly or implicitly qualified <jar id> specified in the <jar name> specified by *TJ*.
4) Let *JR* be the JAR identified by *JN*.
5) Let *P* be the value of the `path` parameter. If *P* does not conform to the format for <SQL Java path>, then an exception condition is raised: *Java DDL — invalid path*.

   NOTE 72 — The `path` parameter can be an empty or all-blank string.
6) The current catalog and schema at the time of the call to the SQLJ.ALTER_JAVA_PATH procedure are the default, respectively, for each omitted <catalog name> and <schema name> in the <resolution jar>s contained in *P*. For each <path element> *PE* (if any) in *P*, *PE*'s <resolution jar> is updated to reflect the defaults for any omitted <catalog name>s and <schema name>s.
7) For each <path element> *PE* (if any) in *P*, let *RJ* be the <jar name> contained in the <resolution jar> contained in *PE*. If *RJ* is equivalent to *JN*, then an exception condition is raised: *Java DDL — self-referencing path*.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8) If *P* cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning — SQL-Java path too long for information schema*.

NOTE 73 — The Information Schema is defined in [ISO9075-11].

9) The value of *P* is placed in the SQL-Java path of the JAR descriptor of *JR*, replacing the current SQL-Java path (if any) associated with *JR*.

10) If an invocation of the SQLJ.ALTER_JAVA_PATH procedure raises an exception condition, then effect on the path associated with the JAR is implementation-defined.

11) The effect of SQLJ.ALTER_JAVA_PATH on SQL statements that have already been prepared or are currently executing is implementation-dependent.

Conformance Rules

1) Without Feature J601, "SQL-Java paths", conforming SQL language shall not contain invocations of the SQLJ.ALTER_JAVA_PATH procedure.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

12 Java topics

12.1 Java facilities supported by this part of ISO/IEC 9075

12.1.1 Package java.sql

SQL systems that implement this part of ISO/IEC 9075 support the package `java.sql`, which is the JDBC driver, and all classes required by that package. The other Java packages supplied by SQL systems thatimplement this part of ISO/IEC 9075 are implementation-defined.

In an SQL system that implements this part of ISO/IEC 9075, the package `java.sql` supports the *default connection*. The default connection for a Java method invoked as an SQL routine has the following characteristics:

— The default connection is pre-allocated to provide efficient access to the database.

— The default connection is included in the current session and transaction.

— The authorization ID of the default connection is the current authorization ID.

— The JDBC AUTOCOMMIT setting of the default connection is *false*.

Other data source URLs that are supported by `java.sql` are implementation-defined.

12.1.2 System properties

SQL systems that implement this part of ISO/IEC 9075 support the following system properties for use by the `getProperty` method of `java.lang.System`:

Table 2 — System properties

Key	Description of associated value
`sqlj.defaultconnection`	If a Java method is executing in an SQL-implementation, then the String "`jdbc:default:connection`"[1]
`sqlj.runtime`	The class name of a runtime context class[2]

[1] Otherwise, the null value.

[2] This class is a subclass of the class `sqlj.runtime.RuntimeContext`. The `getDefaultContext()` method of the class whose name is returned returns the default connection described in Subclause 12.1.1, “Package java.sql”.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

12.2 Deployment descriptor files

Function

Supply information for actions to be taken by the SQLJ.INSTALL_JAR and SQLJ.REMOVE_JAR procedures.

Model

A deployment descriptor file is a text file contained in a JAR, which is specified with the following property in the manifest for the JAR:

```
Name: file_name
SQLJDeploymentDescriptor: TRUE
```

Properties

The text contained in a deployment descriptor file shall have the following form:

```
<descriptor file> ::=
  SQLActions <left bracket> <right bracket> <equals operator>
      { [ <double quote> <action group> <double quote>
      [ <comma> <double quote> <action group> <double quote> ] ] }

<action group> ::=
    <install actions>
  | <remove actions>

<install actions> ::=
  BEGIN INSTALL [ <command> <semicolon> ]... END INSTALL

<remove actions> ::=
  BEGIN REMOVE [ <command> <semicolon> ]... END REMOVE

<command> ::=
    <SQL statement>
  | <implementor block>

<SQL statement> ::=
  !! See Description

<implementor block> ::=
  BEGIN <implementor name> <SQL token>... END <implementor name>

<implementor name> ::=
  <identifier>

<SQL token> ::=
  !! See Description
```

Description

1) <descriptor file> shall contain at most one <install actions> and at most one <remove actions>.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2) The <command>s specified in the <install actions> (if any) and <remove actions> (if any) specify the *install actions* and *remove actions* of the deployment descriptor file, respectively.

3) An <SQL statement> specified in an <install actions> shall be either:

   a) An <SQL-invoked routine> whose <language clause> specifies JAVA. The procedures and functions created by those statements are called the *deployed routines* of the deployment descriptor file.

   b) A <grant privilege statement> that specifies the EXECUTE privilege for a deployed routine.

   c) A <user-defined type definition> that specifies an <external Java type clause>. The types created by those statements are called the *deployed types* of the deployment descriptor file.

   d) A <grant privilege statement> that specifies the USAGE privilege for a deployed type.

   e) A <user-defined ordering definition> that specifies ordering for a deployed type.

4) When a deployment descriptor file is executed by a call of the `SQLJ.INSTALL_JAR` procedure, if the <jar name> of any <external routine name> or an <SQL-invoked routine> in an <install actions> is the <jar name> "`thisjar`", then "`thisjar`" is effectively replaced with the `jar` parameter of the `SQLJ.INSTALL_JAR` procedure for purposes of that execution.

5) An <SQL statement> specified in a <remove actions> shall be either:

   a) A <drop routine statement> for a deployed routine.

   b) A <revoke statement> for the EXECUTE privilege on a deployed routine.

   c) A <drop data type statement> for a deployed type.

   d) A <revoke statement> for the USAGE privilege on a deployed type.

   e) A <drop user-defined ordering statement> that specifies ordering for a deployed type.

6) An <implementor block> specifies implementation-defined install actions (remove actions) when specified in an <install actions> (<remove actions>).

7) An <SQL token> is an SQL lexical unit specified by the term "<token>" in Subclause 5.2, "<token> and <separator>", in [ISO9075-2]. That is, the comments, quotes, and double-quotes in an <implementor block> follow SQL token conventions.

8) An <implementor name> is an implementation-defined SQL identifier. The <implementor name>s specified following the BEGIN and END keywords shall be equivalent.

9) Whether an <implementor block> with a given <implementor name> contained in an <install actions> (<remove actions>) is interpreted as an install action (remove action) is implementation-defined. That is, an implementation may or may not perform install or remove actions specified by some other implementation.

NOTE 74 — The deployment descriptor file format corresponds to the more general notion of a properties file supporting indexed properties. Therefore, the deployment descriptor file can be used by the SQL-implementation to instantiate a Java Bean, with an indexed property, `SQLActions`. An application developer could then customize the resulting Java Bean instance with ordinary Java Bean tools. For example, SQL procedures or permissions could be changed by changing the routine descriptors stored in the `SQLActions` property. The SQL system can then use the customized Java Bean instance to generate a modified version of the deployment descriptor file to use in a revised version of the JAR. Further information regarding Java Beans can be found in *The JavaBeans™ 1.01 Specification*, `http://java.sun.com/products/javabeans/docs/spec.html`.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Conformance Rules

1) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <user-defined type definition>.

2) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <drop data type statement>.

3) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains an <SQL-invoked routine>.

4) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <drop routine statement>.

5) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <user-defined ordering definition>.

6) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <drop user-defined ordering statement>.

7) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <grant privilege statement>.

8) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <revoke statement>.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13 Information Schema

This Clause modifies Clause 5, "Information Schema", in ISO/IEC 9075-11.

13.1 JAR_JAR_USAGE view

Function

Identify each JAR owned by a given user or role on which JARs defined in this catalog are dependent.

Definition

```
CREATE VIEW JAR_JAR_USAGE AS
    SELECT JJU.PATH_JAR_CATALOG, JJU.PATH_JAR_SCHEMA, JJU.PATH_JAR_NAME,
           JAR_CATALOG, JAR_SCHEMA, JAR_NAME
    FROM ( DEFINITION_SCHEMA.JAR_JAR_USAGE AS JJU
        JOIN
           DEFINITION_SCHEMA.JARS AS J
        USING ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME ) )
      JOIN
         DEFINITION_SCHEMA.SCHEMATA AS S
      ON ( ( JJU.PATH_JAR_CATALOG, JJU.PATH_JAR_SCHEMA )
         = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
          OR
            S.SCHEMA_OWNER IN
            ( SELECT ER.ROLE_NAME
              FROM ENABLED_ROLES AS ER ) )
       AND
          JAR_CATALOG =
          ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME AS ISCN ) ;
GRANT SELECT ON TABLE JAR_JAR_USAGE
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature J652, "SQL/JRT Usage tables", conforming SQL language shall not reference INFORMATION_SCHEMA.JAR_JAR_USAGE.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.2 JARS view

Function

Identify the installed JARs defined in this catalog that are accessible to the current user.

Definition

```
CREATE VIEW JARS AS
    SELECT JAR_CATALOG, JAR_SCHEMA, JAR_NAME, JAVA_PATH
    FROM DEFINITION_SCHEMA.JARS
    WHERE ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME, 'JAR' ) IN
          ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
            FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
            WHERE GRANTEE IN
                  ( 'PUBLIC', CURRENT_USER )
                OR
                  GRANTEE IN
                  ( SELECT ROLE_NAME
                    FROM ENABLED_ROLES ) )
       AND
          JAR_CATALOG =
          ( SELECT CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME ) ;
GRANT SELECT ON TABLE JARS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA.JARS.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.3 METHOD_SPECIFICATIONS view

This Subclause modifies Subclause 5.34, "METHOD_SPECIFICATIONS view", in ISO/IEC 9075-11.

Function

Identify the methods in the catalog that are accessible to a given user or role.

Definition

Add columns EXTERNAL_NAME and IS_FIELD in [ISO9075-11] Add ", EXTERNAL_NAME, IS_FIELD" to the end of the outermost <select list> of the <view definition>.

Conformance Rules

1) Insert this CR Without Feature J651, "SQL/JRT Information Schema", conforming SQL language shall not reference INFORMATION_SCHEMA . METHOD_SPECIFICATIONS . EXTERNAL_NAME.

2) Insert this CR Without Feature J651, "SQL/JRT Information Schema", conforming SQL language shall not reference INFORMATION_SCHEMA . METHOD_SPECIFICATIONS . IS_FIELD.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.4 ROUTINE_JAR_USAGE view

Function

Identify the JARs owned by a given user or role on which external Java routines defined in this catalog are dependent.

Definition

```
CREATE VIEW ROUTINE_JAR_USAGE AS
    SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
           RJU.JAR_CATALOG, RJU.JAR_SCHEMA, RJU.JAR_NAME
    FROM ( DEFINITION_SCHEMA.ROUTINE_JAR_USAGE AS RJU
        JOIN
           DEFINITION_SCHEMA.ROUTINES AS R
        USING ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME ) )
      JOIN
         DEFINITION_SCHEMA.SCHEMATA AS S
      ON ( ( RJU.JAR_CATALOG, RJU.JAR_SCHEMA ) =
           ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
          OR
            S.SCHEMA_OWNER IN
            ( SELECT ER.ROLE_NAME
              FROM ENABLED_ROLES AS ER ) )
       AND
          SPECIFIC_CATALOG =
          ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME AS ISCN ) ;
GRANT SELECT ON TABLE ROUTINE_JAR_USAGE
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature J652, “SQL/JRT Usage tables”, conforming SQL language shall not reference INFORMATION_SCHEMA.ROUTINE_JAR_USAGE.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.5 TYPE_JAR_USAGE view

Function

Identify the JARs owned by a given user or role on which external Java types defined in this catalog are dependent.

Definition

```
CREATE VIEW TYPE_JAR_USAGE AS
    SELECT USER_DEFINED_TYPE_CATALOG AS UDT_CATALOG,
           USER_DEFINED_TYPE_SCHEMA AS UDT_SCHEMA,
           USER_DEFINED_TYPE_NAME AS UDT_NAME,
           TJU.JAR_CATALOG, TJU.JAR_SCHEMA, TJU.JAR_NAME
    FROM ( DEFINITION_SCHEMA.TYPE_JAR_USAGE AS TJU
        JOIN
           DEFINITION_SCHEMA.USER_DEFINED_TYPES AS UDT
        USING ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
                USER_DEFINED_TYPE_NAME ) )
      JOIN
         DEFINITION_SCHEMA.SCHEMATA AS S
      ON ( ( TJU.JAR_CATALOG, TJU.JAR_SCHEMA ) =
           ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
          OR
            S.SCHEMA_OWNER IN
            ( SELECT ER.ROLE_NAME
              FROM ENABLED_ROLES AS ER ) )
       AND
          USER_DEFINED_TYPE_CATALOG =
          ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME AS ISCN ) ;
GRANT SELECT ON TABLE TYPE_JAR_USAGE
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature J652, “SQL/JRT Usage tables”, conforming SQL language shall not reference INFORMATION_SCHEMA.TYPE_JAR_USAGE.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.6 USER_DEFINED_TYPES view

This Subclause modifies Subclause 5.72, "USER_DEFINED_TYPES view", in ISO/IEC 9075-11.

Function

Identify the user-defined types defined in this catalog that are accessible to a given user or role.

Definition

Add columns to the USER_DEFINED_TYPES view in [ISO9075-11] Add "`, EXTERNAL_NAME, EXTERNAL_LANGUAGE, JAVA_INTERFACE`" to the end of the outermost <select list> of the <view definition>.

Conformance Rules

1) Insert this CR Without Feature J651, "SQL/JRT Information Schema", conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . EXTERNAL_NAME.

2) Insert this CR Without Feature J651, "SQL/JRT Information Schema", conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . EXTERNAL_LANGUAGE.

3) Insert this CR Without Feature J651, "SQL/JRT Information Schema", conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . JAVA_INTERFACE.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

13.7 Short name views

This Subclause modifies Subclause 5.77, "Short name views", in ISO/IEC 9075-11.

Function

Provide alternative views that use only identifiers that do not require Feature F391, "Long identifiers".

Definition

Replace view METHOD_SPECS in [ISO9075-11]

```
CREATE VIEW METHOD_SPECS
         ( SPECIFIC_CATALOG,    SPECIFIC_SCHEMA,    SPECIFIC_NAME,
           UDT_CATALOG,         UDT_SCHEMA,         UDT_NAME,
           METHOD_NAME,         IS_STATIC,          IS_OVERRIDING,
           IS_CONSTRUCTOR,      DATA_TYPE,          CHAR_MAX_LENGTH,
           CHAR_OCTET_LENGTH,   CHAR_SET_CATALOG,   CHAR_SET_SCHEMA,
           CHARACTER_SET_NAME,  COLLATION_CATALOG,  COLLATION_SCHEMA,
           COLLATION_NAME,      NUMERIC_PRECISION,  NUMERIC_PREC_RADIX,
           NUMERIC_SCALE,       DATETIME_PRECISION, INTERVAL_TYPE,
           INTERVAL_PRECISION,  RETURN_UDT_CATALOG, RETURN_UDT_SCHEMA,
           RETURN_UDT_NAME,     SCOPE_CATALOG,      SCOPE_SCHEMA,
           SCOPE_NAME,          MAX_CARDINALITY,    DTD_IDENTIFIER,
           METHOD_LANGUAGE,     PARAMETER_STYLE,    IS_DETERMINISTIC,
           SQL_DATA_ACCESS,     IS_NULL_CALL,       TO_SQL_SPEC_CAT,
           TO_SQL_SPEC_SCHEMA,  TO_SQL_SPEC_NAME,   AS_LOCATOR,
           EXTERNAL_NAME,       IS_FIELD,           CREATED,
           LAST_ALTERED ) AS
    SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
           USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME,
           METHOD_NAME, IS_STATIC, IS_OVERRIDING,
           IS_CONSTRUCTOR, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH,
           CHARACTER_OCTET_LENGTH, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
           CHARACTER_SET_NAME, COLLATION_CATALOG, COLLATION_SCHEMA,
           COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
           NUMERIC_SCALE, DATETIME_PRECISION, INTERVAL_TYPE,
           INTERVAL_PRECISION, RETURN_UDT_CATALOG, RETURN_UDT_SCHEMA,
           RETURN_UDT_NAME, SCOPE_CATALOG, SCOPE_SCHEMA,
           SCOPE_NAME, MAXIMUM_CARDINALITY, DTD_IDENTIFIER,
           METHOD_LANGUAGE, PARAMETER_STYLE, IS_DETERMINISTIC,
           SQL_DATA_ACCESS, IS_NULL_CALL, TO_SQL_SPECIFIC_CATALOG,
           TO_SQL_SPECIFIC_SCHEMA, TO_SQL_SPECIFIC_NAME, AS_LOCATOR,
           EXTERNAL_NAME, IS_FIELD, CREATED,
           LAST_ALTERED
    FROM INFORMATION_SCHEMA.METHOD_SPECIFICATIONS;
```

Replace view UDT_S in [ISO9075-11]

```
CREATE VIEW UDT_S
         ( UDT_CATALOG,         UDT_SCHEMA,         UDT_NAME,
           UDT_CATEGORY,        IS_INSTANTIABLE,    IS_FINAL,
           ORDERING_FORM,       ORDERING_CATEGORY,  ORDERING_ROUT_CAT,
           ORDERING_ROUT_SCH,   ORDERING_ROUT_NAME, REFERENCE_TYPE,
```

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        DATA_TYPE,          CHAR_MAX_LENGTH,    CHAR_OCTET_LENGTH,
        CHAR_SET_CATALOG,   CHAR_SET_SCHEMA,    CHARACTER_SET_NAME,
        COLLATION_CATALOG,  COLLATION_SCHEMA,   COLLATION_NAME,
        NUMERIC_PRECISION,  NUMERIC_PREC_RADIX, NUMERIC_SCALE,
        DATETIME_PRECISION, INTERVAL_TYPE,      INTERVAL_PRECISION,
        SOURCE_DTD_ID,      REF_DTD_IDENTIFIER, EXTERNAL_NAME,
        EXTERNAL_LANGUAGE,  JAVA_INTERFACE ) AS
SELECT USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, USER_DEFINED_TYPE_NAME,
        CATEGORY, IS_INSTANTIABLE, IS_FINAL,
        ORDERING_FORM, ORDERING_CATEGORY, ORDERING_ROUTINE_CATALOG,
        ORDERING_ROUTINE_SCHEMA, ORDERING_ROUTINE_NAME, REFERENCE_TYPE,
        DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
        CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
        COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME,
        NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
        DATETIME_PRECISION, INTERVAL_TYPE, INTERVAL_PRECISION,
        SOURCE_DTD_IDENTIFIER, REF_DTD_IDENTIFIER, EXTERNAL_NAME,
        EXTERNAL_LANGUAGE, JAVA_INTERFACE
FROM INFORMATION_SCHEMA.USER_DEFINED_TYPES;
```

Conformance Rules

1) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . METHOD_SPECS . EXTERNAL_NAME.

2) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . METHOD_SPECS . IS_FIELD.

3) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . EXTERNAL_NAME.

4) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . EXTERNAL_LANGUAGE.

5) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . JAVA_INTERFACE.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14 Definition Schema

This Clause modifies Clause 6, "Definition Schema", in ISO/IEC 9075-11.

14.1 JAR_JAR_USAGE base table

Function

The JAR_JAR_USAGE table has one row for each JAR included in the SQL-Java path of a JAR.

Definition

```
CREATE TABLE JAR_JAR_USAGE (
    JAR_CATALOG            INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_SCHEMA             INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_NAME               INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PATH_JAR_CATALOG       INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PATH_JAR_SCHEMA        INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PATH_JAR_NAME          INFORMATION_SCHEMA.SQL_IDENTIFIER,
    CONSTRAINT JAR_JAR_USAGE_PRIMARY_KEY
      PRIMARY KEY ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME,
                    PATH_JAR_CATALOG, PATH_JAR_SCHEMA, PATH_JAR_NAME ),
    CONSTRAINT JAR_JAR_USAGE_CHECK_REFERENCES_JARS
      CHECK ( PATH_JAR_CATALOG NOT IN
              ( SELECT CATALOG_NAME
                FROM SCHEMATA )
            OR
              ( PATH_JAR_CATALOG, PATH_JAR_SCHEMA, PATH_JAR_NAME ) IN
              ( SELECT JAR_CATALOG, JAR_SCHEMA, JAR_NAME
                FROM JARS ) ),
    CONSTRAINT JAR_JAR_USAGE_FOREIGN_KEY_JARS
      FOREIGN KEY ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME )
        REFERENCES JARS
    )
```

Description

1) The JAR_JAR_USAGE table has one row for each JAR *JP* identified by a <jar name> contained in an <SQL Java path> associated with JAR *J*.

2) The values of JAR_CATALOG, JAR_SCHEMA, and JAR_NAME are the <catalog name>, <unqualified schema name>, and <jar id>, respectively, of the <jar name> of the JAR (*J*) being described.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

3) The values of PATH_JAR_CATALOG, PATH_JAR _SCHEMA, and PATH_JAR _NAME are the <catalog name>, <unqualified schema name>, and <jar id>, respectively, of the <jar name> of a JAR (*JP*) that is in the <SQL Java path> associated with JAR *J*.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.2 JARS base table

Function

The JARS table has one row for each installed JAR.

Definition

```
CREATE TABLE JARS (
    JAR_CATALOG            INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_SCHEMA             INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_NAME               INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAVA_PATH              INFORMATION_SCHEMA.CHARACTER_DATA,
    CONSTRAINT JARS_PRIMARY_KEY
      PRIMARY KEY ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME ),
    CONSTRAINT JAR_FOREIGN_KEY_SCHEMATA
      FOREIGN KEY ( JAR_CATALOG, JAR_SCHEMA )
        REFERENCES SCHEMATA
    )
```

Description

1) The values of JAR_CATALOG, JAR_SCHEMA, and JAR_NAME are the <catalog name>, <unqualified schema name>, and <jar id> of the <jar name> of the JAR being described.

2) Case

   a) If the character representation of the SQL-Java path in the descriptor of the JAR being described can be represented without truncation, then the value of JAVA_PATH is that character representation.

   b) Otherwise, the value of JAVA_PATH is the null value.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.3 METHOD_SPECIFICATIONS base table

This Subclause modifies Subclause 6.30, "METHOD_SPECIFICATIONS base table", in ISO/IEC 9075-11.

Function

The METHOD_SPECIFICATIONS base table has one row for each method specification.

Definition

Replace CONSTRAINT METHOD_SPECIFICATIONS_LANGUAGE_CHECK in [ISO9075-11]

```
CONSTRAINT METHOD_SPECIFICATIONS_LANGUAGE_CHECK
  CHECK ( METHOD_LANGUAGE IN
  ( 'SQL', 'ADA', 'C', 'COBOL',
    'FORTRAN', 'MUMPS', 'PASCAL','PLI', 'JAVA' ) )
```

Add two columns and three constraints in [ISO9075-11] Add the following <table element>s to the <table element list> of the METHOD_SPECIFICATIONS base table:

```
EXTERNAL_NAME          INFORMATION_SCHEMA.CHARACTER_DATA,
IS_FIELD               INFORMATION_SCHEMA.CHARACTER_DATA
CONSTRAINT METHOD_SPECIFICATIONS_IS_FIELD_CHECK
  CHECK ( IS_FIELD IN ( 'YES', 'NO' ) ),
CONSTRAINT METHOD_SPECIFICATIONS_METHOD_COMBINATIONS
  CHECK ( ( ( METHOD_LANGUAGE = 'JAVA' )
          AND
             ( EXTERNAL_NAME, IS_FIELD ) IS NOT NULL )
         OR
           ( ( METHOD_LANGUAGE <> 'JAVA' )
          AND
             ( ( EXTERNAL_NAME, IS_FIELD )
             IS NULL ) ) ),
CONSTRAINT METHOD_SPECIFICATIONS_FIELD_COMBINATIONS
  CHECK ( IS_FIELD = 'NO' OR IS_STATIC = 'YES' )
```

Description

1) Insert this Desc. Case:

   a) If the method being described is an external Java routine, then the value of EXTERNAL_NAME is the <Java method and parameter declarations> specified in the <external Java method clause> for that external Java data type.

   b) If the method being described is a static field of an external Java type, then the value of EXTERNAL_NAME is the <qualified Java field name> specified in the <static field method spec> of the method.

   c) Otherwise, the value of EXTERNAL_NAME is the null value.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

2) Insert this Desc. Case:

a) If the method being described is a static field of an external Java type, then the value of IS_FIELD is 'YES'.

b) If the method being described is an external Java type, then the value of IS_FIELD is 'NO'.

c) Otherwise, the value of IS_FIELD is the null value.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.4 ROUTINE_JAR_USAGE base table

Function

The ROUTINE_JAR_USAGE table has one row for each external Java routine that names a JAR in an <external Java reference string>.

Definition

```
CREATE TABLE ROUTINE_JAR_USAGE (
    SPECIFIC_CATALOG       INFORMATION_SCHEMA.SQL_IDENTIFIER,
    SPECIFIC_SCHEMA        INFORMATION_SCHEMA.SQL_IDENTIFIER,
    SPECIFIC_NAME          INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_CATALOG            INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_SCHEMA             INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_NAME               INFORMATION_SCHEMA.SQL_IDENTIFIER,
    CONSTRAINT ROUTINE_JAR_USAGE_PRIMARY_KEY
      PRIMARY KEY ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
                    JAR_CATALOG, JAR_SCHEMA, JAR_NAME ),
    CONSTRAINT JAR_JAR_USAGE_CHECK_REFERENCES_JARS
      CHECK ( JAR_CATALOG NOT IN
              ( SELECT CATALOG_NAME
                FROM SCHEMATA )
            OR
              ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME ) IN
              ( SELECT JAR_CATALOG, JAR_SCHEMA, JAR_NAME
                FROM JARS ) ),
    CONSTRAINT JAR_JAR_USAGE_FOREIGN_KEY_ROUTINES
      FOREIGN KEY (SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME )
        REFERENCES ROUTINES
    )
```

Description

1) The ROUTINE_JAR_USAGE table has one row for each external Java routine that names a JAR in an <external Java reference string>.

2) The values of SPECIFIC_CATALOG, SPECIFIC_SCHEMA, and SPECIFIC_NAME are the <catalog name>, <unqualified schema name>, and <qualified identifier>, respectively, of the <specific name> of the external Java routine being described.

3) The values of JAR_CATALOG, JAR_SCHEMA, and JAR_NAME are the <catalog name>, <unqualified schema name>, and <jar id>, respectively, of the <jar name> of the JAR being referenced in the external Java routine's <external Java reference string>.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.5 ROUTINES base table

This Subclause modifies Subclause 6.40, "ROUTINES base table", in ISO/IEC 9075-11.

Function

The ROUTINES table has one row for each SQL-invoked routine.

Definition

Replace CONSTRAINT EXTERNAL_LANGUAGE_CHECK in [ISO9075-11] Add "`, 'JAVA'`" to the <in value list> of valid EXTERNAL_LANGUAGE values.

Description

No additional Descriptions.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.6 TYPE_JAR_USAGE base table

Function

The TYPE_JAR_USAGE table has one row for each external Java type.

Definition

```
CREATE TABLE TYPE_JAR_USAGE (
    USER_DEFINED_TYPE_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
    USER_DEFINED_TYPE_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
    USER_DEFINED_TYPE_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_CATALOG                  INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_SCHEMA                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
    JAR_NAME                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
    CONSTRAINT TYPE_JAR_USAGE_PRIMARY_KEY
      PRIMARY KEY (USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
                   USER_DEFINED_TYPE_NAME, JAR_CATALOG, JAR_SCHEMA, JAR_NAME),
    CONSTRAINT TYPE_JAR_USAGE_CHECK_REFERENCES_JARS
      CHECK ( JAR_CATALOG NOT IN
              ( SELECT CATALOG_NAME
                FROM SCHEMATA )
            OR
              ( JAR_CATALOG, JAR_SCHEMA, JAR_NAME ) IN
              ( SELECT JAR_CATALOG, JAR_SCHEMA, JAR_NAME
                FROM JARS ) ),
    CONSTRAINT TYPE_JAR_USAGE_FOREIGN_KEY_USER_DEFINED_TYPES
      FOREIGN KEY (USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
                   USER_DEFINED_TYPE_NAME ) REFERENCES USER_DEFINED_TYPES
    )
```

Description

1) The TYPE_JAR_USAGE table has one row for each external Java type.

2) The values of USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA, and USER_DEFINED_TYPE_NAME are the <catalog name>, <unqualified schema name>, and <qualified identifier>, respectively, of the <user-defined type name> of the external Java type being described.

3) The values of JAR_CATALOG, JAR_SCHEMA, and JAR_NAME are the <catalog name>, <unqualified schema name>, and <jar id>, respectively, of the <jar name> of the JAR being referenced in the external Java type's <external Java class clause>.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

14.7 USAGE_PRIVILEGES base table

This Subclause modifies Subclause 6.59, "USAGE_PRIVILEGES base table", in ISO/IEC 9075-11.

Function

The USAGE_PRIVILEGES table has one row for each usage privilege descriptor. It effectively contains a representation of the usage privilege descriptors.

Definition

Replace CONSTRAINT USAGE_PRIVILEGES_OBJECT_TYPE_CHECK in [ISO9075-11] Add "`, 'JAR'`" to the <in value list> of valid OBJECT_TYPE values.

Replace CONSTRAINT USAGE_PRIVILEGES_CHECK_REFERENCES_OBJECT in [ISO9075-11] Add the following to the end of the <query expression> contained in the <in predicate>:

```
UNION
  SELECT JAR_CATALOG, JAR_SCHEMA, JAR_NAME, 'JAR'
  FROM JARS
```

Description

1) Augment Desc. 4)

JAR	The object to which the privilege applies is a JAR.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

14.8 USER_DEFINED_TYPES base table

This Subclause modifies Subclause 6.61, "USER_DEFINED_TYPES base table", in ISO/IEC 9075-11.

Function

The USER_DEFINED_TYPES table has one row for each user-defined type.

Definition

Add three columns to the USER_DEFINED_TYPES base table in [ISO9075-11]

Add CONSTRAINT USER_DEFINED_TYPES_COMBINATIONS in [ISO9075-11]

Add the following <table element>s to the <table element list> of the USER_DEFINED_TYPES base table:

```
EXTERNAL_NAME           INFORMATION_SCHEMA.CHARACTER_DATA,
EXTERNAL_LANGUAGE       INFORMATION_SCHEMA.CHARACTER_DATA
  CONSTRAINT USER_DEFINED_TYPE_EXTERNAL_LANGUAGE_CHECK
    CHECK ( EXTERNAL_LANGUAGE IN ( 'JAVA' ) ),
JAVA_INTERFACE          INFORMATION_SCHEMA.CHARACTER_DATA
  CONSTRAINT USER_DEFINED_TYPE_JAVA_INTERFACE_CHECK
    CHECK ( JAVA_INTERFACE IN ( 'SERIALIZABLE', 'SQLDATA' ) ),
CONSTRAINT USER_DEFINED_TYPES_COMBINATIONS
  CHECK ( ( ( EXTERNAL_LANGUAGE = 'JAVA' ) AND
            ( EXTERNAL_NAME, JAVA_INTERFACE ) IS NOT NULL )
       OR
         ( ( EXTERNAL_LANGUAGE, EXTERNAL_NAME, JAVA_INTERFACE )
           IS NULL ) )
```

Augment CONSTRAINT USER_DEFINED_TYPES_ORDERING_CATEGORY_CHECK in [ISO9075-11]

Add "`, 'COMPARABLE'`" to the <in value list> of valid ORDERING_CATEGORY values.

Description

1) Augment Desc. 7)

COMPARABLE	Two values of this type may be compared with `java.lang.Comparable`'s `compareTo( )` method.

2) Insert this Desc. Case:

   a) If the user-defined type being described is an external Java data type, then the value of EXTERNAL_NAME is the <jar and class name> specified in the <external Java class clause> for that external Java data type.

   b) Otherwise, the value of EXTERNAL_NAME is the null value.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3) Insert this Desc. Case:

   a) If the user-defined type being described is an external Java data type, then the value of EXTERNAL_LANGUAGE is 'JAVA'.

   b) Otherwise, the value of EXTERNAL_LANGUAGE is the null value.

4) Insert this Desc. Case:

   a) If the user-defined type being described is an external Java data type, then the value of JAVA_INTERFACE is the <interface specification> specified for that external Java data type.

   b) Otherwise, the value of JAVA_INTERFACE is the null value.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

15 Status codes

This Clause modifies Clause 24, "Status codes", in ISO/IEC 9075-2.

15.1 SQLSTATE

This Subclause modifies Subclause 24.1, "SQLSTATE", in ISO/IEC 9075-2.

Augment Table 33, "SQLSTATE class and subclass values"

Table 3 — SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
	All alternatives from ISO/IEC 9075-2			
X	*Java DDL*[1]	46	(*no subclass*)	000
			invalid URL	001
			invalid JAR name	002
			invalid class deletion	003
			invalid replacement	005
			attempt to replace uninstalled JAR	00A
			attempt to remove uninstalled JAR	00B
			invalid JAR removal	00C
			invalid path	00D
			self-referencing path	00E
X	*Java execution*[1]	46	(*no subclass*)	000
			invalid JAR name in path	102
			unresolved class name	103
W	*warning*	01	(*no subclass*)	000

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Category	Condition	Class	Subcondition	Subclass
			SQL-Java path too long for information schema	011

[1] The Condition names "*Java DDL*" and "*Java execution*" are given the same Class code given to Condition name "*OLB-specific error*" in [ISO9075-10]; there is no conflict with Subcondition values for the Class code.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16 Conformance

16.1 Claims of conformance to SQL/JRT

In addition to the requirements of ISO/IEC 9075-1, Clause 8, "Conformance", a claim of conformance to this part of ISO/IEC 9075 shall:

1) Claim conformance to at least one of:
   - Feature J621, "external Java routines"
   - Feature J541, "SERIALIZABLE"
   - Feature J551, "SQLDATA"
2) Claim conformance to at least one of:
   - Feature J511, "Commands"
   - Feature J531, "Deployment"

16.2 Additional conformance requirements for SQL/JRT

Each claim of conformance to one of the following features:

- Feature J621, "external Java routines"
- Feature J622, "external Java types"
- Feature J561, "JAR privileges"

shall state that Feature J511, "Commands", or Feature J531, "Deployment", or both support it.

16.3 Implied feature relationships of SQL/JRT

Table 4 — Implied feature relationships of SQL/JRT

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
J541	SERIALIZABLE	J622	external Java types

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

16.3 Implied feature relationships of SQL/JRT

Feature ID	Feature Name	Implied Feature ID	Implied Feature Name
J551	SQLDATA	J622	external Java types

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex A
(informative)

SQL Conformance Summary

This Annex modifies Annex A, "SQL Conformance Summary", in ISO/IEC 9075-2.

The contents of this Annex summarizes all Conformance Rules, ordered by Feature ID and by Subclause.

1) Specifications for Feature J511, "Commands":

   a) Subclause 9.4, "<user-defined type definition>":

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

   b) Subclause 9.7, "<drop data type statement>":

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

   c) Subclause 9.8, "<SQL-invoked routine>":

      i) Insert this CR Without Feature J511, "Commands", conforming SQL language shall not contain an <SQL-invoked routine> that contains a <language name> that contains JAVA and that is not contained in a <descriptor file>.

   d) Subclause 9.10, "<drop routine statement>":

      i) Insert this CR Without Feature J511, "Commands", conforming SQL language shall not contain a <drop routine statement> that contains a <specific routine designator> that identifies an external Java routine and that is not contained in a <descriptor file>.

   e) Subclause 9.11, "<user-defined ordering definition>":

      i) Insert this CR Without Feature J511, "Commands", conforming SQL language shall not contain a <user-defined ordering definition> that contains a <schema-resolved user-defined type name> that identifies an external Java type and that is not contained in a <descriptor file>.

   f) Subclause 9.12, "<drop user-defined ordering statement>":

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

   g) Subclause 10.1, "<grant privilege statement>":

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

i) Insert this CR Without Feature J511, "Commands", conforming SQL language shall not contain a <grant privilege statement> that contains an <object name> that immediately contains a <jar name> and that is not contained in a <descriptor file>.

h) Subclause 10.3, "<revoke statement>":

i) Insert this CR Without Feature J511, "Commands", conforming SQL language shall not contain a <revoke statement> that an <object name> that immediately contains a <jar name> and that is not contained in a <descriptor file>.

2) Specifications for Feature J521, "JDBC data types":

a) Subclause 9.8, "<SQL-invoked routine>":

i) Insert this CR Without Feature J521, "JDBC data types", conforming SQL language shall not contain a <Java data type> that is not the corresponding Java data type of some SQL data type.

3) Specifications for Feature J531, "Deployment":

a) Subclause 11.1, "SQLJ.INSTALL_JAR procedure":

i) Without Feature J531, "Deployment", conforming SQL language shall not contain invocations of the SQLJ.INSTALL_JAR procedure that provide non-zero values of the `deploy` parameter.

b) Subclause 11.3, "SQLJ.REMOVE_JAR procedure":

i) Without Feature J531, "Deployment", conforming SQL language shall not contain invocations of the SQLJ.REMOVE_JAR procedure that provide non-zero values of the `undeploy` parameter.

c) Subclause 12.2, "Deployment descriptor files":

i) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <user-defined type definition>.

ii) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <drop data type statement>.

iii) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains an <SQL-invoked routine>.

iv) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <drop routine statement>.

v) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <user-defined ordering definition>.

vi) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <drop user-defined ordering statement>.

vii) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <grant privilege statement>.

viii) Without Feature J531, "Deployment", conforming SQL language shall not contain an <SQL statement> that contains a <revoke statement>.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

4) Specifications for Feature J541, "SERIALIZABLE":

   a) Subclause 9.4, "<user-defined type definition>":

      i) Insert this CR Without Feature J541, "SERIALIZABLE", conforming SQL language shall not contain an <interface specification> that contains SERIALIZABLE.

5) Specifications for Feature J551, "SQLDATA":

   a) Subclause 9.4, "<user-defined type definition>":

      i) Insert this CR Without Feature J551, "SQLDATA", conforming SQL language shall not contain an <interface specification> that contains SQLDATA.

6) Specifications for Feature J561, "JAR privileges":

   a) Subclause 10.2, "<privileges>":

      i) Insert this CR Without Feature J561, "JAR privileges", conforming SQL language shall not contain an <object name> that immediately contains a <jar name>.

7) Specifications for Feature J571, "NEW operator":

   a) Subclause 6.2, "<new specification>":

      i) Insert this 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.

8) Specifications for Feature J581, "Output parameters":

   a) Subclause 9.8, "<SQL-invoked routine>":

      i) Insert this CR Without Feature J581, "Output parameters", conforming SQL language shall not contain an <SQL-invoked routine> that contains a <language name> that contains JAVA and that contains a <parameter mode> that contains either OUT or INOUT.

9) Specifications for Feature J591, "Overloading":

   a) Subclause 9.4, "<user-defined type definition>":

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

10) Specifications for Feature J601, "SQL-Java paths":

   a) Subclause 8.2, "<SQL Java path>":

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

   b) Subclause 11.4, "SQLJ.ALTER_JAVA_PATH procedure":

      i) Without Feature J601, "SQL-Java paths", conforming SQL language shall not contain invocations of the SQLJ.ALTER_JAVA_PATH procedure.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

11) Specifications for Feature J611, “References”:

a) Subclause 8.3, “<routine invocation>”:

i) Insert this CR Without Feature J611, “References”, conforming SQL language shall not contain a <reference value expression>.

ii) Insert this CR Without Feature J611, “References”, conforming SQL language shall not contain a <right arrow>.

12) Specifications for Feature J621, “external Java routines”:

a) Subclause 9.8, “<SQL-invoked routine>”:

i) Insert this CR Without Feature J621, “external Java routines”, conforming SQL language shall not contain an <SQL-invoked routine> that contains a <language name> that contains JAVA.

b) Subclause 9.10, “<drop routine statement>”:

i) Insert this CR Without Feature J621, “external Java routines”, conforming SQL language shall not contain a <drop routine statement> that contains a <specific routine designator> that identifies an external Java routine.

13) Specifications for Feature J622, “external Java types”:

a) Subclause 9.4, “<user-defined type definition>”:

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

b) Subclause 9.7, “<drop data type statement>”:

i) Insert this CR Without Feature J622, “external Java types”, conforming SQL language shall not contain a <drop data type statement> that contains a <schema-resolved user-defined type name> that identifies an external Java type.

c) Subclause 9.11, “<user-defined ordering definition>”:

i) Insert this CR Without Feature J622, “external Java types”, conforming SQL language shall not contain a <user-defined ordering definition> that contains a <schema-resolved user-defined type name> that identifies an external Java type.

d) Subclause 9.12, “<drop user-defined ordering statement>”:

i) Insert this CR Without Feature J622, “external Java types”, conforming SQL language shall not contain a <drop user-defined ordering statement> that contains a <schema-resolved user-defined type name> that identifies an external Java type.

14) Specifications for Feature J631, “Java signatures”:

a) Subclause 8.1, “<Java parameter declaration list>”:

i) Without Feature J631, “Java signatures”, conforming SQL language shall not contain a <Java parameter declaration list> that is not equivalent to the default Java method signature as determined in Subclause 8.6, “Java routine signature determination”.

15) Specifications for Feature J641, “Static fields”:

a) Subclause 9.4, “<user-defined type definition>”:

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

i) Insert this CR Without Feature J641, “Static fields”, conforming SQL language shall not contain a <static field method spec>.

16) Specifications for Feature J651, “SQL/JRT Information Schema”:

a) Subclause 13.2, “JARS view”:

i) Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA.JARS.

b) Subclause 13.3, “METHOD_SPECIFICATIONS view”:

i) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . METHOD_SPECIFICATIONS . EXTERNAL_NAME.

ii) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . METHOD_SPECIFICATIONS . IS_FIELD.

c) Subclause 13.6, “USER_DEFINED_TYPES view”:

i) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . EXTERNAL_NAME.

ii) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . EXTERNAL_LANGUAGE.

iii) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . JAVA_INTERFACE.

d) Subclause 13.7, “Short name views”:

i) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . METHOD_SPECS . EXTERNAL_NAME.

ii) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . METHOD_SPECS . IS_FIELD.

iii) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . EXTERNAL_NAME.

iv) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . EXTERNAL_LANGUAGE.

v) Insert this CR Without Feature J651, “SQL/JRT Information Schema”, conforming SQL language shall not reference INFORMATION_SCHEMA . UDT_S . JAVA_INTERFACE.

17) Specifications for Feature J652, “SQL/JRT Usage tables”:

a) Subclause 13.1, “JAR_JAR_USAGE view”:

i) Without Feature J652, “SQL/JRT Usage tables”, conforming SQL language shall not reference INFORMATION_SCHEMA.JAR_JAR_USAGE.

b) Subclause 13.4, “ROUTINE_JAR_USAGE view”:

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

i) Without Feature J652, “SQL/JRT Usage tables”, conforming SQL language shall not reference INFORMATION_SCHEMA.ROUTINE_JAR_USAGE.

c) Subclause 13.5, “TYPE_JAR_USAGE view”:

i) Without Feature J652, “SQL/JRT Usage tables”, conforming SQL language shall not reference INFORMATION_SCHEMA.TYPE_JAR_USAGE.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex B
(informative)

Implementation-defined elements

This Annex modifies Annex B, "Implementation-defined elements", in ISO/IEC 9075-2.

This Annex references those features that are identified in the body of this part of ISO/IEC 9075 as implementation-defined.

1) Subclause 4.8.5, "Converting objects between SQL and Java":

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

2) Subclause 4.10, "Privileges":

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

      NOTE 75 — This is similar to the implementation-defined privileges required for creating a schema.

   b) Invocations of Java methods referenced by SQL names are governed by the normal EXECUTE privilege on SQL routine names. It is implementation-defined 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.

3) Subclause 4.11.1, "Deployment descriptor files":

   a) An implementation-defined implementor block can be provided in a deployment descriptor file to allow specification of custom install and remove actions.

4) Subclause 5.2, "Names and identifiers":

   a) The character set supported, and the maximum lengths of the <package identifier>, <class identifier>, <Java field name>, and <Java method name> are implementation-defined.

5) Subclause 6.2, "<new specification>":

   a) 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.

6) Subclause 8.3, "<routine invocation>":

   a) If validation of the <Java parameter declaration list> has been implementation-defined to be performed by <routine invocation>, then the Syntax Rules of Subclause 8.6, "Java routine signature determination", are applied with <routine invocation> as *ELEMENT*, 0 (zero) as *INDEX*, and *SR* as *SUBJECT*.

   b) For an external Java routine, let CPV_i be an implementation-defined non-null value of declared type T_i.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

c) The method of execution of a subject Java class's implementation of `writeObject()` to convert a Java value to an SQL value is implementation-defined.

d) The method of execution of a subject Java class's implementation of `writeSQL()` to convert a Java value to an SQL value is implementation-defined.

e) The method of execution of a subject Java class's implementation of `readObject()` to convert an SQL value to a Java object is implementation-defined.

f) The method of execution of a subject Java class's implementation of `readSQL()` to convert an SQL value to a Java object is implementation-defined.

g) If *R* is an external Java routine, then if the JDBC connection object that created any element of *RS* is closed, then the effect is implementation-defined.

h) If *R* is an external Java routine, 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.

i) If *R* is an external Java routine, then whether the call of *P* returns update counts as defined in JDBC is implementation-defined.

7) Subclause 9.4, "<user-defined type definition>":

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

b) If *UDT* is an external Java data type, then it is implementation-defined whether validation of the explicit or implicit <Java parameter declaration list> is performed by <user-defined type definition> or when the corresponding SQL-invoked method is invoked.

8) Subclause 9.5, "<attribute definition>":

a) The method of execution of a subject Java class's implementation of `writeObject()` to convert a Java value to an SQL value is implementation-defined.

b) The method of execution of a subject Java class's implementation of `writeSQL()` to convert a Java value to an SQL value is implementation-defined.

c) The method of execution of a subject Java class's implementation of `readObject()` to convert an SQL value to a Java object is implementation-defined.

d) The method of execution of a subject Java class's implementation of `readSQL()` to convert an SQL value to a Java object is implementation-defined.

9) Subclause 9.8, "<SQL-invoked routine>":

a) The maximum value of <maximum returned result sets> is implementation-defined.

b) If *R* is an external Java routine, then it is implementation-defined whether validation of the explicit or implicit <Java parameter declaration list> is performed by <SQL-invoked routine> or when its SQL-invoked routine is invoked.

10) Subclause 11.1, "SQLJ.INSTALL_JAR procedure":

a) The maximum length for the CHARACTER VARYING parameters is an implementation-defined integer value.

b) The privileges required to invoke the `SQLJ.INSTALL_JAR` procedure are implementation-defined.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

c) The SQLJ.INSTALL_JAR procedure is subject to implementation-defined rules for executing SQL-schema statements within SQL-transactions.

d) If an invocation of SQLJ.INSTALL_JAR raises an exception condition, then the effect on the install actions is implementation-defined.

e) The values of the url parameter that are valid are implementation-defined, and may include URLs whose format is implementation-defined. If the value of the url parameter does not conform to implementation-defined restrictions and does not identify a valid JAR, then an exception condition is raised: *Java DDL — invalid URL*.

11) Subclause 11.2, "SQLJ.REPLACE_JAR procedure":

a) The maximum length for the CHARACTER VARYING parameters is an implementation-defined integer value.

b) The privileges required to invoke the SQLJ.REPLACE_JAR procedure are implementation-defined.

c) The SQLJ.REPLACE_JAR procedure is subject to implementation-defined rules for executing SQL-schema statements within SQL-transactions.

d) The values of the url parameter that are valid are implementation-defined, and may include URLs whose format is implementation-defined. If the value of the url parameter does not conform to implementation-defined restrictions and does not identify a valid JAR, then an exception condition is raised: *Java DDL — invalid URL*.

12) Subclause 11.3, "SQLJ.REMOVE_JAR procedure":

a) The maximum length for the CHARACTER VARYING parameters is an implementation-defined integer value.

b) The privileges required to invoke the SQLJ.REMOVE_JAR procedure are implementation-defined.

c) The SQLJ.REMOVE_JAR procedure is subject to implementation-defined rules for executing SQL-schema statements within SQL-transactions.

d) If an invocation of SQLJ.REMOVE_JAR raises an exception condition, then the effect on the remove actions is implementation-defined.

13) Subclause 11.4, "SQLJ.ALTER_JAVA_PATH procedure":

a) The maximum length for the CHARACTER VARYING parameters is an implementation-defined integer value.

b) The privileges required to invoke the SQLJ.ALTER_JAVA_PATH procedure are implementation-defined.

c) The SQLJ.ALTER_JAVA_PATH procedure is subject to implementation-defined rules for executing SQL-schema statements within SQL-transactions.

d) If an invocation of the SQLJ.ALTER_JAVA_PATH procedure raises an exception condition, then effect on the path associated with the JAR is implementation-defined.

14) Subclause 12.1.1, "Package java.sql":

a) SQL systems that implement this part of ISO/IEC 9075 support the package java.sql, which is the JDBC driver, and all classes required by that package. The other Java packages supplied by SQL systems that implement this part of ISO/IEC 9075 are implementation-defined.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

b) In an SQL system that implements this part of ISO/IEC 9075, the package `java.sql` supports the default connection. Other data source URLs that are supported by `java.sql` are implementation-defined.

15) Subclause 12.2, "Deployment descriptor files":

a) An <implementor name> is an implementation-defined SQL identifier.

b) An <implementor block> specifies implementation-defined install actions and remove actions when specified in <install actions> and <remove actions>, respectively.

c) Whether an <implementor block> with a given <implementor name> contained in an <install actions> (<remove actions>) is interpreted as an install action (remove action) is implementation-defined. That is, an implementation may or may not perform install or remove actions specified by some other implementation.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex C
(informative)

Implementation-dependent elements

This Annex modifies Annex C, "Implementation-dependent elements", in ISO/IEC 9075-2.

This Annex references those places where this part of ISO/IEC 9075 states explicitly that the actions of a conforming implementation are implementation-dependent.

1) Subclause 3.2.1, "Specification of built-in procedures":

   a) The manner in which built-in procedures are defined is implementation-dependent.

2) Subclause 4.8, "User-defined types":

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

3) Subclause 8.3, "<routine invocation>":

   a) 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.

4) Subclause 11.2, "SQLJ.REPLACE_JAR procedure":

   a) The effect of `SQLJ.REPLACE_JAR` on currently executing SQL statements that use an SQL routine or structured type whose implementation has been replaced is implementation-dependent.

5) Subclause 11.3, "SQLJ.REMOVE_JAR procedure":

   a) The effect of `SQLJ.REMOVE_JAR` on currently executing SQL statements that use an SQL routine or structured type whose implementation has been removed is implementation-dependent.

6) Subclause 11.4, "SQLJ.ALTER_JAVA_PATH procedure":

   a) The effect of `SQLJ.ALTER_JAVA_PATH` on SQL statements that have already been prepared or are currently executing is implementation-dependent.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex D
(infomative)

Deprecated features

This Annex modifies Annex D, "Deprecated features", in ISO/IEC 9075-2.

It is intended that the following features will be removed at a later date from a revised version of this part of ISO/IEC 9075:

None.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex E
(informative)

Incompatibilities with ISO/IEC 9075:2003

This Annex modifies Annex E, "Incompatibilities with ISO/IEC 9075:2003", in ISO/IEC 9075-2.

This part of ISO/IEC 9075 introduces some incompatibilities with the earlier version of Database Language SQL as specified in ISO/IEC 9075:2003. Unless specified in this Annex, features and capabilities of Database Language SQL are compatible with the earlier version of ISO/IEC 9075:2003.

None.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex F
(informative)

SQL feature taxonomy

This Annex modifies Annex F, "SQL feature taxonomy", in ISO/IEC 9075-2.

This Annex describes a taxonomy of features defined in this part of ISO/IEC 9075.

Table 5, "Feature taxonomy for optional features", contains a taxonomy of the features of the SQL language that are specified in this part of ISO/IEC 9075.

In this table, the first column contains a counter that may be used to quickly locate rows of the table; these values otherwise have no use and are not stable — that is, they are subject to change in future editions of or even Technical Corrigenda to ISO/IEC 9075 without notice.

The "Feature ID" column of this table specifies the formal identification of each feature and each subfeature contained in the table.

The "Feature Name" column of this table contains a brief description of the feature or subfeature associated with the Feature ID value.

Table 5, "Feature taxonomy for optional features", does not provide definitions of the features; the definition of those features is found in the Conformance Rules that are further summarized in Annex A, "SQL Conformance Summary".

Table 5 — Feature taxonomy for optional features

	Feature ID	Feature Name
1	**J511**	**Commands**
2	**J521**	**JDBC data types**
3	**J531**	**Deployment**
4	**J541**	**SERIALIZABLE**
5	**J551**	**SQLDATA**
6	**J561**	**JAR privileges**
7	**J571**	**NEW operator**
8	**J581**	**Output parameters**

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	Feature ID	Feature Name
9	J591	Overloading
10	J601	SQL-Java paths
11	J611	References
12	J621	external Java routines
13	J622	external Java types
14	J631	Java signatures
15	J641	Static fields
16	J651	SQL/JRT Information Schema
17	J652	SQL/JRT Usage tables

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Annex G
(informative)

Defect reports not addressed in this edition of this part of ISO/IEC 9075

Each entry in this Annex describes a reported defect in the previous edition of this part of ISO/IEC 9075 that remains in this edition.

None.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex H
(informative)

Routines tutorial

H.1 Technical components

This part of ISO/IEC 9075 includes the following:

— New built-in procedures.

- `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.

— New built-in schema.

  The built-in schema named `SQLJ` is assumed to be in all catalogs of an SQL system that implements the SQL/JRT facility, and to contain all of the built-in procedures of the SQL/JRT facility.

— Extensions of the following SQL statements:

- CREATE PROCEDURE/FUNCTION — to specify an SQL name for a Java method.
- DROP PROCEDURE/FUNCTION — to delete the SQL name of a Java method.
- CREATE TYPE — to specify an SQL name for a Java class.
- DROP TYPE — to delete the SQL name of a Java class.
- GRANT — to grant the USAGE privilege on Java JARs.
- REVOKE — to revoke the USAGE privilege on Java JARs.

— Conventions for returning values ofOUT and INOUT parameters, and for returning SQL result sets.

— New forms of reference: Qualified references to the fields and methods of columns whose data types are defined on Java classes.

— Additional views and columns in the Information Schema.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

H.2 Overview

This tutorial shows a series of example Java classes, indicates how they can be installed, and shows how their static, public methods can be referenced with SQL/JRT facilities in an SQL-environment.

The example Java methods assume an SQL table named EMPS, with the following columns:

— NAME — the employee's name.

— ID — the employee's identification.

— STATE — the state in which the employee is located.

— SALES — the amount of the employee's sales.

— JOBCODE — the job code of the employee.

The table definition is:

```
CREATE TABLE emps (
    name    VARCHAR(50),
    id      CHARACTER(5),
    state   CHARACTER(20),
    sales   DECIMAL (6,2),
    jobcode INTEGER );
```

The example classes and methods are:

— `Routines1.region` — A Java method that maps a US state code to a region number. This method doesn't use SQL internally.

— `Routines1.correctStates` — A Java method that performs an SQL UPDATE statement to correct the spelling of *state* codes. The old and new spellings are specified by input-mode parameters.

— `Routines2.bestTwoEmps` — A Java method that determines the top two employees by their sales, and returns the columns of those two employee rows as output-mode parameter values. This method creates an SQL result set and processes it internally.

— `Routines3.orderedEmps` — A Java method that creates an SQL result set consisting of selected employee rows ordered by the sales column, and returns that result set to the client.

— `Over1.isOdd` and `Over2.isOdd` — Contrived Java methods to illustrate overloading rules.

— `Routines4.job1` and `Routines5.job2` — Java methods that return a string value corresponding to an integer jobcode value. These methods illustrate the treatment of null arguments.

— `Routines6.job3` — Another Java method that returns a string value corresponding to an integer jobcode value. This method illustrates the behavior of static Java variables.

Unless otherwise noted, the methods that invoke SQL use JDBC. One of the methods is shown in both aversion using JDBC and a version using SQL/OLB. The others could also be coded with SQL/OLB.

It is assumed that the import statements `import java.sql.*;` and `java.math.*;` have been included in all classes.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

H.3 Example Java methods: region and correctStates

This clause shows an example Java class, `Routines1`, with two simple methods.

— The `int`-valued static method `region` categorizes 9 states into 3 geographic regions, returning an integer indicating the region associated with a valid state or throwing an exception for invalid states. This method will be called as a function in SQL.

— The `void` method `correctStates` updates the EMPS table to correct spelling errors in the state column. This method will be called as a procedure in SQL.

```
public class Routines1 {
  //An int method that will be called as a function
  public static int region(String s) throws SQLException {
    if (s.equals("MN") || s.equals("VT") || s.equals("NH")) return 1;
      else if (s.equals("FL") || s.equals("GA") || s.equals("AL")) return 2;
      else if (s.equals("CA") || s.equals("AZ") || s.equals("NV")) return 3;
      else throw new SQLException("Invalid state code", "38001");
  }
  //A void method that will be called as a stored procedure
  public static void correctStates (String oldSpelling, String newSpelling)
      throws SQLException {
    Connection conn = DriverManager.getConnection ("jdbc:default:connection");
    PreparedStatement stmt = conn.prepareStatement
        ("UPDATE emps SET state = ? WHERE state = ?");
    stmt.setString(1, newSpelling);
    stmt.setString(2, oldSpelling);
    stmt.executeUpdate();
    stmt.close();
    conn.close();
    return;
  }
}
```

H.4 Installing region and correctStates in SQL

The source code for Java classes such as `Routines1` will normally be in one or more Java files (*i.e.*, files with file type “java”). When you compile them (using the `javac` compile command), the resulting code will be in one or more class files (*i.e.*, files with file type “class”). You then typically collect a set of class files into a Java JAR, which is a ZIP-coded collection of files.

To use Java classes in SQL, you load a JAR containing them into the SQL system by calling the SQL `SQLJ.INSTALL_JAR` procedure. The `SQLJ.INSTALL_JAR` procedure is a new built-in SQL procedure that makes the collection ofJava classes contained in a specified JAR available for use in the current SQL catalog. For example, assume that you have assembled the above `Routines1` class into a JAR with local file name “`~/classes/Routines1.jar`”:

```
SQLJ.INSTALL_JAR('file:~/classes/Routines1.jar', 'routines1_jar', 0)
```

— The first parameter of the `SQLJ.INSTALL_JAR` procedure is a character string specifying the URL of the given JAR. This parameter is never folded to upper case.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— The second parameter of the SQLJ.INSTALL_JAR procedure is a character string that will be used as the name of the JAR in the SQL system. The JAR name is an SQL qualified name, and follows SQL conventions for qualified names.

  The JAR name that you specify as the second parameter of the SQLJ.INSTALL_JAR procedure identifies the JAR within the SQL system. That is, the JAR name that you specify is used only in SQL, and has nothing to do with the contents of the JAR itself. The JAR name is used in the following contexts, which are described in later clauses:

  - As a parameter of the SQLJ.REMOVE_JAR and SQLJ.REPLACE_JAR procedures.
  - As a qualifier of Java class names in SQL CREATE PROCEDURE/FUNCTION statements.
  - As an operand of the extended SQL GRANT and REVOKE statements.
  - As a qualifier of Java class names in SQL CREATE TYPE statements.

  The JAR name may also be used in follow-on facilities for downloading JARs from the SQL system.

— JARs can also contain *deployment descriptors*, which specify implicit actions to be taken by the SQLJ.INSTALL_JAR and SQLJ.REMOVE_JAR procedures. The third parameter of the SQLJ.INSTALL_JAR procedureis an integer that specifies whether you do or do not (indicated by non-zero or zero values, respectively) want the SQLJ.INSTALL_JAR procedure to execute the actions specified by a deployment descriptor in the JAR. Deployment descriptors are further described in Subclause 12.2, "Deployment descriptor files".

The name of the INSTALL_JAR procedure is qualified with the schema name SQLJ. All built-in procedures of the SQL/JRT facility are defined to be contained in that built-in schema. The SQLJ schema is assumed to be present in each catalog of an SQL system that implements the SQL/JRT facility.

The first two parameters of SQLJ.INSTALL_JAR are character strings, so if you specify them as literals, you will use single quotes, not the double quotes used for SQL delimited identifiers.

The actions of the SQLJ.INSTALL_JAR procedure are as follows:

— Obtain the JAR designated by the first parameter.

— Extract the class files that it contains and install them into the current SQL schema.

— Retain a copy of the JAR itself, and associate it with the value of the second parameter.

— If the third parameter is non-zero, then perform the actions specified by the deployment descriptor of the JAR.

After you install a JAR with the SQLJ.INSTALL_JAR procedure, you can reference the static methods of the classes contained in that JAR in the CREATE PROCEDURE/FUNCTION statement, as we will describe in the next Subclause.

H.5 Defining SQL names for region and correctStates

Before you can call a Java method in SQL, you shall define an SQL name for it. You do this with new options on the SQL CREATE PROCEDURE/FUNCTION statement. For example:

```
CREATE PROCEDURE correct_states(old CHARACTER(20), new CHARACTER(20))
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  MODIFIES SQL DATA
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines1_jar:Routines1.correctStates';
CREATE FUNCTION region_of(state CHARACTER(20)) RETURNS INTEGER
  NO SQL
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines1_jar:Routines1.region';
```

The CREATE PROCEDURE and CREATE FUNCTION statements specify SQL names and Java method signatures for the Java methods specified in the EXTERNAL NAME clauses. The format of the method names in the external name clause consists of the JAR name that was specified in the `SQLJ.INSTALL_JAR` procedure followed by the Java method name, fully qualified with the package name(s) (if any) and class name.

The CREATE PROCEDURE for `correct_states` specifies the clause MODIFIES SQL DATA. This indicates that the specified Java method modifies (via INSERT, UPDATE, or DELETE) data in SQL tables. The CREATE FUNCTION for `region_of` specifies NO SQL. This indicates that the specified Java method performs no SQL operations.

Other clauses that you can specify are READS SQL DATA, which indicates that the specified Java method reads (through SELECT) data in SQL tables, but does not modify SQL data, and CONTAINS SQL, which indicates that the specified method invokes SQL operations, but neither reads nor modifies SQL data. The alternative CONTAINS SQL is the default.

You use the SQL procedure and function names that you define with such CREATE PROCEDURE/FUNCTION statements as normal SQL procedure and function names:

```
SELECT name, region_of(state) AS region
FROM emps
WHERE region_of(state) = 3;
CALL correct_states ('GEO', 'GA');
```

You can define multiple SQL names for the same Java method:

```
CREATE PROCEDURE state_correction(old CHARACTER(20), new CHARACTER(20))
  MODIFIES SQL DATA
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines1_jar:Routines1.correctStates';
CREATE FUNCTION state_region(state CHARACTER(20)) RETURNS INTEGER
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines1_jar:Routines1.region';
```

The various SQL function and procedure names for a Java method can be used equivalently:

```
SELECT name, state_region(state) AS region
FROM emps
WHERE region_of(state) = 2;
CALL state_correction ('ORE', 'OR');
```

The SQL names are normal 3-part SQL names, and the first two parts of the 3-part names are defaulted as defined in SQL for CREATE PROCEDURE and CREATE FUNCTION statements.

There are other considerations for the CREATE PROCEDURE/FUNCTION statement, dealing with parameter data types, overloaded names, and privileges, which we will discuss in later Subclauses.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

H.6 A Java method with output parameters: bestTwoEmps

The parameters of the `region` and `correctStates` methods are all input-only parameters. This is the normal Java parameter convention.

SQL procedures also support parameters with mode OUT and INOUT. The Java language does not directly have a notion of output parameters. SQL/JRT therefore uses arrays to return output values for parameters of Java methods. That is, if you want an `Integer` parameter to return a value to the caller, you specify the type of that parameter to be `Integer[ ]`, *i.e.* an array of `Integer`. Such an array will contain only one element: the input value of the parameter is contained in that element when the method is called, and the method sets the value of that element to the desired output value.

As we will see in the following clauses, this use of arrays for output parameters in the Java methods is visible only to the Java method. When you call such a method as an SQL procedure, you supply normal scalar data items as parameters. The SQL system performs the mapping between those scalar data items and Java arrays implicitly.

The following Java method illustrates the way that you code output parameters in Java. This method, `bestTwoEmps`, returns the `name`, `id`, `region`, and `sales` of the two employees that have the highest `sales` in the regions with numbers higher than a parameter value. That is, each of the first 8 parameters is an OUT parameter, and is therefore declared to be an array of the given type.

The following version of the `bestTwoEmps` method uses SQL/OLB for statements that access SQL:

```
public class Routines2 {
  public static void bestTwoEmps (
      String[ ] n1, String[ ] id1, int[ ] r1, BigDecimal[ ] s1,
      String[ ] n2, String[ ] id2, int[ ] r2, BigDecimal[ ] s2,
      int regionParm) throws SQLException {
    #sql iterator ByNames (String name, String id, int region, BigDecimal sales);
    n1[0]= "****"; n2[0]= "****"; id1[0]= ""; id2[0]= "";
    r1[0]=0; r2[0]=0; s1[0]= new BigDecimal(0); s2[0]= new BigDecimal(0);
    ByNames r;
    try {
      #sql r = {SELECT name, id, region_of(state) AS region, sales
                FROM emp
                WHERE region_of(state) > :regionParm
                  AND sales IS NOT NULL
                ORDER BY sales DESC};
      if (r.next()) {
        n1[0] = r.name();
        id1[0] = r.id();
        r1[0] = r.region();
        s1[0] = r.sales();
      }
      else return;
      if (r.next()) {
        n2[0] = r.name();
        id2[0] = r.id();
        r2[0] = r.region();
        s2[0] = r.sales();
      }
      elsereturn;
    } finally r.close();
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  }
}
```

Note that since the above Java method uses SQL/OLB for SQL operations, it does not have to explicitly obtain a connection to the SQL system. By default, SQL/OLB executes any SQL contained in a routine in the context of the SQL statement invoking that routine.

For comparison, here's a version of the `bestTwoEmps` method using JDBC instead of SQL/OLB:

```
public class Routines2 {
  public static void bestTwoEmps (
      String[ ] n1, String[ ] id1, int[ ] r1, BigDecimal[ ] s1,
      String[ ] n2, String[ ] id2, int[ ] r2, BigDecimal[ ] s2,
      int regionParm) throws SQLException {
    n1[0]= "****"; n2[0]= "****"; id1[0]= ""; id2[0]= "";
    r1[0]=0; r2[0]=0; s1[0]= new BigDecimal(0); s2[0]= new BigDecimal(0);
    try {
      Connection conn = DriverManager.getConnection
          ("jdbc:default:connection");
      java.sql.PreparedStatement stmt = conn.prepareStatement
          ("SELECT name, id, region_of(state) AS region, sales
           FROM emp
           WHERE region_of(state) > ?
             AND sales IS NOT NULL
           ORDER BY sales DESC");
      stmt.setInt(1, regionParm)
      ResultSet r = stmt.executeQuery();
      if (r.next()) {
        n1[0] = r.getString("name");
        id1[0] = r.getString("id");
        r1[0] = r.getInt("region");
        s1[0] = r.getBigDecimal("sales");
      }
      elsereturn;
      if (r.next()) {
        n2[0] = r.getString("name");
        id2[0] = r.getString("id");
        r2[0] = r.getInt("region");
        s2[0] = r.getBigDecimal("sales");
      }
      else return;
    } finally { stmt.close() };
  }
}
```

H.7 A CREATE PROCEDURE best2 for bestTwoEmps

Assume that you call the `SQLJ.INSTALL_JAR` procedure for a JAR containing the `Routines2` class with the `bestTwoEmps` method:

```
SQLJ.INSTALL_JAR ('file:~/classes/Routines2.jar', 'routines2_jar', 0)
```

As indicated previously, in order to call a method such as `bestTwoEmps` in SQL, you shall define an SQL name for it, using the CREATE PROCEDURE statement:

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
CREATE PROCEDURE best2 (
    OUT n1 CHARACTER VARYING(50), OUT id1 CHARACTER VARYING(5), OUT r1 INTEGER,
    OUT s1 DECIMAL(6,2),
    OUT n2 CHARACTER VARYING(50), OUT id2 CHARACTER VARYING(5), OUT r2 INTEGER,
    OUT s2 DECIMAL(6,2), region INTEGER)
  READS SQL DATA
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines2_jar:Routines2.bestTwoEmps';
```

For parameters that are specified to be OUT or INOUT, the corresponding Java parameter shall be an array of the corresponding data type.

H.8 Calling the best2 procedure

After you have installed the `Routines2` class in an SQL system and executed the CREATE PROCEDURE for `best2`, you can call the `bestTwoEmps` method as if it were an SQL stored procedure, with normal conventions for OUT parameters. Such a call could be written with embedded SQL, CLI, ODBC, or JDBC. The following is an example of such a call using JDBC:

```
java.sql.CallableStatement stmt = conn.prepareCall(
    "{call best2(?,?,?,?,?,?,?,?,?)}");
  stmt.registerOutParameter(1, java.sql.Types.STRING);
  stmt.registerOutParameter(2, java.sql.Types.STRING);
  stmt.registerOutParameter(3, java.sql.Types.INTEGER);
  stmt.registerOutParameter(4, java.sql.Types.DECIMAL);
  stmt.registerOutParameter(5, java.sql.Types.STRING);
  stmt.registerOutParameter(6, java.sql.Types.STRING);
  stmt.registerOutParameter(7, java.sql.Types.INTEGER);
  stmt.registerOutParameter(8, java.sql.Types.DECIMAL);
  stmt.setInt(9, 3);
  stmt.executeUpdate();
  String n1 = stmt.getString(1);
  String id1 = stmt.getString(2);
  int r1 = stmt.getInt(3);
  BigDecimal s1 = stmt.getBigDecimal(4);
  String n2 = stmt.getString(5);
  String id2 = stmt.getString(6);
  int r2 = stmt.getInt(7);
  BigDecimal s2 = stmt.getBigDecimal(8);
```

H.9 A Java method returning a result set: orderedEmps

SQL stored procedures can generate SQL result sets as their output. An SQL result set (as defined in JDBC and SQL) is an ordered sequence of SQL rows. SQL result sets aren't processed as normal function result values, but are instead bound to caller-specified iterators or cursors, which are subsequently used to process the rows of the result set.

The following Java method, `orderedEmps`, generates an SQL result setand then returns that result set to the client. Note that the `orderedEmps` method internally generates the result set in the same way as the

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

`bestTwoEmps` method. However, the `bestTwoEmps` method processes the result set within the `bestTwoEmps` method itself, whereas this `orderedEmps` method returns the result set to the client as an SQL result set.

To write a Java method that returns a result set to the client, you specify the method to have an additional parameter that is a single-element array of either the Java `ResultSet` class or a class generated by an SQL/OLB iterator declaration ("`#sql iterator...`").

The following version of the `orderedEmps` procedure uses SQL/OLB to access the SQL server, and returns the result set as an SQL/OLB iterator, `SalesReport`:

```
// #sql public iterator SalesReport (String name, int region, BigDecimal sales);
  public class Routines3 {
    public static void orderedEmps (int regionParm, SalesReport[ ] rs)
        throws SQLException {
      #sql rs[0] = { SELECT name, region_of(state) AS region, sales
                     FROM emp
                     WHERE region_of(state) > :regionParm
                       AND sales IS NOT NULL
                     ORDER BY sales DESC };
      return;
    }
  }
```

The `SalesReport` iterator class could be a public static inner class of `Routines3`. However, the above example presumes existence of an "`*.sqlj`" file, named `SalesReport.sqlj`, in the same package as `Routines3`, containing the public definition of the `SalesReport` iterator. That is, `SalesReport.sqlj` contains:

```
#sql public iterator SalesReport (String name, int region, BigDecimal sales);
```

Assume, for this example, that both class `Routines3` and the iterator `SalesReport` are defined in a package named `classes`.

For comparison, the following shows `orderedEmps` written using JDBC instead of SQL/OLB.

```
public class Routines3 {
  public static void orderedEmps(int regionParm, ResultSet[ ] rs)
      throws SQLException {
    Connection conn = DriverManager.getConnection ("jdbc:default:connection");
    java.sql.PreparedStatement stmt = conn.prepareStatement
        ("SELECT name, region_of(state) AS region, sales
          FROM emp WHERE region_of(state) > ?
           AND sales IS NOT NULL
          ORDER BY sales DESC");
    stmt.setInt (1, regionParm);
    rs[0] = stmt.executeQuery();
    return;
  }
}
```

The method sets the first element of the `ResultSet[]` parameter to reference the Java `ResultSet` containing the SQL result set to be returned. The method does *not* close either the returned `ResultSet` object *or* the Java statement object that generated the result set. The SQL system will implicitly close both of those objects.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

You can call a method such as `orderedEmps` in Java in the normal manner, supplying explicit arguments for both parameters. You can also call it in SQL, as a stored procedure that generates a result set to be processed in the SQL manner. We illustrate how this is done in the following two clauses.

Each of the above `orderedEmps` examples has a single result set parameter, `rs`, in which you can only return a single result set. You can also specify multiple result set parameters. See Subclause 9.8, "<SQL-invoked routine>".

Note that, in comparison to the prior examples of `bestTwoEmps`, there is no `try...finally` block to close the SQL/OLB iterator or `ResultSet`, `rs[0]`, or the JDBC PreparedStatement, `stmt`. For a result set to be returned from a stored procedure it shall not be explicitly closed, which means, in the case of JDBC, that the statement executed to generate the result set also shall not be explicitly closed.

H.10 A CREATE PROCEDURE rankedEmps for orderedEmps

Assume that you call the `SQLJ.INSTALL_JAR` procedure for a JAR containing the `Routines3` class with the `orderedEmps` method:

```
SQLJ.INSTALL_JAR( 'file:~/classes/Routines3.jar', 'routines3_jar', 0)
```

As with previous methods, you will now need to define an SQL name for the `orderedEmps` method before you can call it as an SQL procedure. As above, you will do this with a CREATE PROCEDURE statement that specifies an EXTERNAL...LANGUAGE JAVA clause to reference the `orderedEmps` method. The following is an example CREATE PROCEDURE...DYNAMIC RESULT SETS for the above `orderedEmps` method:

```
CREATE PROCEDURE ranked_emps (region INTEGER)
    READS SQL DATA
    DYNAMIC RESULT SETS 1
    LANGUAGE JAVA PARAMETER STYLE JAVA
    EXTERNAL NAME 'routines3_jar:classes.Routines3.orderedEmps';
```

A CREATE PROCEDURE statement for a Java method that generates SQL result sets has the following characteristics:

— The DYNAMIC RESULT SETS clause indicates that the procedure generates one or more result sets. The integer specified in the DYNAMIC RESULT SETS clause is the maximum number of result sets that the procedure will generate. If an execution generates more than this number of result sets, a warning will be issued, and only the specified number of result sets will be returned.

— The SQL signature specifies only the parameters that the caller explicitly supplies.

— The specified Java method actually has one or more additional, trailing parameters, whose data types shall be a Java array of either `java.sql.ResultSet` or an implementation of `sqlj.runtime.ResultSetIterator`.

The above CREATE PROCEDURE statement could be used to reference either an SQL/OLB-based or JDBC-based version of `Routines3.orderedEmps`. When it is necessary to choose a particular implementation, the Java method signature of the desired Java method shall be explicitly stated. For the SQL/OLB-based `orderedEmps`:

```
CREATE PROCEDURE ranked_emps (region INTEGER)
    READS SQL DATA
    DYNAMIC RESULT SETS 1
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    LANGUAGE JAVA PARAMETER STYLE JAVA
    EXTERNAL NAME
        'routines3_jar:classes.Routines3.orderedEmps(int, classes.SalesReport[])';
```

And, for the JDBC-based orderedEmps:

```
CREATE PROCEDURE ranked_emps (region INTEGER)
    READS SQL DATA
    DYNAMIC RESULT SETS 1
    LANGUAGE JAVA PARAMETER STYLE JAVA
    EXTERNAL NAME
        'routines3_jar:classes.Routines3.orderedEmps(int, java.sql.ResultSet[])';
```

The only difference in the above CREATE PROCEDURE `ranked_emps` statements is in the Java method signature's description of the dynamic result set returned. In both cases, a fully qualified class name is provided for, respectively, the SQL/OLB iterator (remember that `SalesReport` is in the package named `classes`) and the JDBC result set.

The next clause will show an example invocation of this procedure.

H.11 Calling the rankedEmps procedure

After you have installed the `Routines3` class in an SQL system and executed the CREATE PROCEDURE for `rankedEmps`, you can call the `rankedEmps` procedure as if it were an SQL stored procedure. Such a call could be written with any facility that defines mechanisms for processing SQL result sets — that is, SQL/CLI, JDBC, and SQL/OLB. The following is an example of such a call using JDBC:

```
java.sql.CallableStatement stmt = conn.prepareCall( "{call ranked_emps(?)}");
  stmt.setInt(1, 3);
  ResultSet rs = stmt.executeQuery();
  while (rs.next()) {
    String name = rs.getString(1);
    int region = rs.getInt(2);
    BigDecimal sales = rs.getBigDecimal(3);
    System.out.print("Name = " + name);
    System.out.print("Region = " + region);
    System.out.print("Sales = " + sales);
    System.out.println();
}
```

Note that the call of the `ranked_emps` procedure supplies only the single parameter that was declared in the CREATE PROCEDURE statement. The SQL system then implicitly supplies, as applicable, a parameter that is an empty array of `ResultSet` or an empty array of `classes.SalesReport`, and calls the Java method. That Java method assigns the output result set or iterator to the array parameter. And, when the Java method completes, the SQL system returns the result set or iterator in that output array element as an SQL result set.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

H.12 Overloading Java method names and SQL names

When you use CREATE PROCEDURE/FUNCTION statements to specify SQL names for Java methods, the SQL names can be overloaded. That is, you can specify the same SQL name in multiple CREATE PROCEDURE/FUNCTION statements. Note that support for such SQL overloading is an optional feature.

Consider the following Java classes and methods. These are contrived routines intended only to illustrate overloading, and we won't show the routine bodies.

```
public class Over1 {
  public static int isOdd (int i) {...};
  public static int isOdd (float f) {...};
  public static int testOdd (double d) {...};
}
public class Over2 {
  public static int isOdd (java.sql.Timestamp t) {...};
  public static int oddDateTime (java.sql.Date d) {...};
  public static int oddDateTime (java.sql.Time t) {...};
}
```

Note that the `isOdd` method name is overloaded in the `Over1` class, and the `oddDateTime` method name is overloaded in the `Over2` class.

Assume that the above classes are in a JAR `~/classes/Over.jar`, which you install:

```
SQLJ.INSTALL_JAR ('file:~/classes/Over.jar', 'over_jar', 0)
```

To reference these methods in SQL, you will of course need to specify SQL names for them with CREATE FUNCTION statements. These CREATE FUNCTION statements can specify SQL names that are overloaded. The overloading of the SQL names is completely separate from the overloading in the Java names. This is illustrated in the following.

Recall that you can specify the same Java method in multiple CREATE PROCEDURE/FUNCTION statements.

```
CREATE FUNCTION odd (INTEGER) RETURNS INTEGER
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'over_jar:Over1.isOdd';
CREATE FUNCTION odd (REAL) RETURNS INTEGER
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'over_jar:Over1.isOdd';
CREATE FUNCTION odd (DOUBLE PRECISION) RETURNS INTEGER
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'over_jar:Over1.testOdd';
CREATE FUNCTION odd (TIMESTAMP) RETURNS INTEGER
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'over_jar:Over2.isOdd';
CREATE FUNCTION odd (DATE) RETURNS INTEGER
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'over_jar:Over2.oddDateTime';
CREATE FUNCTION odd (TIME) RETURNS INTEGER
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'over_jar:Over2.oddDateTime';
CREATE FUNCTION is_odd (INTEGER) RETURNS INTEGER
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'over_jar:Over1.isOdd';
CREATE FUNCTION test_odd (REAL) RETURNS INTEGER
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'over_jar:Over1.isOdd';
```

Note the following characteristics of these CREATE FUNCTION statements:

— The SQL name `odd` is defined on the two `isOdd` methods and the `testOdd` method of `Over1`, and also the `isOdd` method and two `oddDateTime` methods of `Over2`. That is, the SQL name `odd` *spans* both overloaded and non-overloaded Java names.

— The SQL names `is_odd` and `test_odd` are defined on the two `isOdd` methods of `Over1`. That is, those two different SQL names are defined on the same Java name.

The rules governing overloading are those of the SQL language as defined in Subclause 11.51, "<SQL-invoked routine>", in [ISO9075-2], and in Subclause 10.4, "<routine invocation>", in [ISO9075-2]. This includes:

— Rules governing what parameter combinations can be overloaded. That is, the legality (or not) of the following CREATE statements is determined by SQL language rules:

```
  CREATE FUNCTION is_odd (INTEGER) RETURNS INTEGER...
  CREATE FUNCTION is_odd (SMALLINT) RETURNS INTEGER...
  CREATE PROCEDURE is_odd (SMALLINT) ...
```

— Rules governing the resolution of calls using overloaded SQL names. That is, the determination of which Java method is called by "`odd(x)`" for some data item "`x`" is determined by SQL language rules.

The EXTERNAL NAME clauses of the above CREATE FUNCTION statements specify only the JAR name and method name of the Java method. For example:

```
CREATE FUNCTION odd (INTEGER) RETURNS INTEGER
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'over_jar:Over1.isOdd';
```

You can also include the Java method signature (*i.e.*, a list of the parameter data types) of a method in the EXTERNAL NAME clause. For example:

```
CREATE FUNCTION odd (INTEGER) RETURNS INTEGER
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'over_jar:Over1.isOdd (int)';
```

The group of eight example CREATE FUNCTION statements, shown earlier in this clause, do not require Java method signatures, but you can include them for clarity. Subclause H.14, "Java method signatures in the CREATE statements", describes cases where the Java method signature is required.

H.13 Java main methods

[Java] places special no requirements on any method named `main`. However, a JVM recognizes a method named `main`, with the following Java method signature, as the method to invoke when only a class name is provided:

```
public static void main (String[ ]);
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

If you specify a Java method named main in an SQL CREATE PROCEDURE...EXTERNAL statement, then that Java method shall have the above Java method signature. The signature of the SQL procedure can either be:

— A single parameter that is an array of CHARACTER or CHARACTER VARYING. That array is passed to the Java method as the String array parameter. Note: This SQL method signature can only be used in SQL systems that support array data types in SQL.

— Zero or more parameters, each of which is CHARACTER or CHARACTER VARYING. Those *N* parameters are passed to the Java method as a single *N* element array of String.

H.14 Java method signatures in the CREATE statements

Consider the following method, `job1`, which has an integer parameter and returns a `String` with the job corresponding with a jobcode value:

```
public class Routines4 {
  //A String method that will be called as a function
  public static String job1 (Integer jc) throws SQLException {
    if (jc == 1) return "Admin";
    else if (jc == 2) return "Sales";
    else if (jc == 3) return "Clerk";
else if (jc == null) return null;
    else return "unknown jobcode";
  }
}
```

Note thatwe suffix the method name with a “1” in anticipation of subsequent variants of the method.

Assume that you install this class in SQL:

```
SQLJ.INSTALL_JAR ('file:~/classes/Routines4.jar', 'routines4_jar', 0)
```

You might want to specify an SQL function `job_of1` defined on the `job1` method:

```
CREATE FUNCTION job_of1(jc INTEGER) RETURNS CHARACTER VARYING(20)
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines4_jar:Routines4.job1';
```

However, as written above, this CREATE statement is not valid. Note that the data type of the parameter of the Java method `job1` is Java `Integer` (which is short for `java.lang.Integer`), and we have specified the SQL data type INTEGER for the corresponding parameter of the SQL `job_of1` function. However, the detailed rules (see Subclause 9.8, “<SQL-invoked routine>” for the external Java form of the SQL CREATE PROCEDURE/FUNCTION statement specifies that the default Java parameter data type for an SQL INTEGER parameter is the Java `int` data type, not the Java `Integer` data type. Subclause H.15, “Null argument values and the RETURNS NULL clause”, describes some reasons why you may want to specify Java `Integer` rather than Java `int`.

If you want to specify an SQL CREATE PROCEDURE/FUNCTION statement for a Java method whose parameter data types include Java types differing from their default Java types, then you specify those data types in a Java method signature in the CREATE statement. This Java method signature is written after the Java method name in the EXTERNAL NAME clause. For example, the above CREATE statement for the `job1` method would be written as:

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
CREATE FUNCTION job_of1(jc INTEGER) RETURNS CHARACTER VARYING(20)
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines4_jar:Routines4.job1(java.lang.Integer)';
```

If you specify data types in the Java method signature of a CREATE statement that specifies DYNAMIC RESULT SETS, then you shall include the implicit trailing result set or iterator parameters in that Java method signature. You do not, however, include those trailing parameters in the SQL signature. For example, you would write the CREATE of Subclause H.10, "A CREATE PROCEDURE rankedEmps for orderedEmps", as follows:

```
CREATE PROCEDURE ranked_emps (region INTEGER)
  READS SQL DATA
  DYNAMIC RESULT SETS 1
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines3_jar:Routines3.orderedEmps (int, java.sql.ResultSet[ ]);
```

See Subclause 9.8, "<SQL-invoked routine>".

H.15 Null argument values and the RETURNS NULL clause

Consider the Java method `job1` and the corresponding SQL function `job_of1`, which we defined in Subclause H.14, "Java method signatures in the CREATE statements".

You can call the SQL function `job_of1` in SQL statements such as the following:

```
SELECT name, job_of1(jobcode)
FROM emps
WHERE job_of1(jobcode) <> 'Admin';
```

Suppose that a row of the EMPS table has a null value in the JOBCODE column. Note that the Java data type of the parameter of the `job1` method is Java `Integer` (that is, `java.lang.Integer`). The Java `Integer` data type is a class, rather than a scalar data type, so its values include both numeric values, and also the null reference value. When an SQL null value is passed as an argument to a Java parameter whose data type is a Java class, the null SQL value is passed as a Java null reference. Such a null reference can be tested within the Java method, as shown in `Routines4.job1`.

Now consider the following similar method, which specifies its parameter data type to be the Java scalar data type `int`, rather than the Java class `Integer`.

```
public class Routines5 {
  //A String method that will be called as a function
  public static String job2 (int jc)
      throws SQLException {
    if (jc == 1) return "Admin";
    else if (jc == 2) return "Sales";
    else if (jc == 3) return "Clerk";
    else return "unknown jobcode";
  }
}
```

Assume that you install this class in SQL:

```
SQLJ.INSTALL_JAR( 'file:~/classes/Routines5.jar', 'routines5_jar', 0)
CREATE FUNCTION job_of2 (jc INTEGER) RETURNS CHARACTER VARYING(20)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines5_jar:Routines5.job2';
```

You can then call the SQL function `job_of2` in SQL statements such as the following:

```
SELECT name, job_of2 (jobcode)
FROM emps
WHERE job_of2(jobcode) <> 'Admin';
```

When this SELECT statement encounters a row of the EMPS table in which the JOBCODE column is null, the effect of the null value on the call(s) of the `job_of2` function is different than for the previous `job_of` function. The `job_of2` function is defined on the method `Routines5.job2`, whose parameter has the scalar data type `int`, rather than the class data type `java.lang.Integer`. The Java `int` data type (and other Java scalar data types) has no null reference value, and no other representation of a null value. Therefore, if the `job2` method is invoked with a null SQL value, then an exception condition is raised.

To summarize:

— The following Java data types have null reference values, and can accommodate SQL arguments that are null:

  `java.lang.String`, `java.math.BigDecimal`, `byte[ ]`, `java.sql.Date`, `java.sql.Time`, `java.sql.Timestamp`, `java.lang.Double`, `java.lang.Float`, `java.lang.Integer`, `java.lang.Short`, `java.lang.Long`, `java.lang.Boolean`

— The following Java data types are scalar data types that cannot accommodate nulls. An exception condition will be raised if an argument value passed as such a parameter data type is null:

  `boolean`, `byte`, `short`, `int`, `long`, `float`, `double`

The exception condition that is raised when you attempt to pass a null argument to a Java parameter that is a non-nullable data type is analogous to the traditional SQL exception condition that is raised when you attempt to FETCH or SELECT a null column value into a host variable for which you did not specify a null indicator variable. In both cases, the “receiving” parameter or variable is unable to accommodate the actual run-time null value, so an exception condition is raised.

When you code Java methods specifically for use in SQL, you will probably tend to specify Java parameter data types that are the nullable Java data types. You may, however, also want to use Java methods in SQL that were not coded for use in SQL, and that are more likely to specify Java parameter data types that are the scalar (non-nullable) Java data types.

You can call such functions in contexts where null values will occur by invoking them conditionally, *e.g.*, in CASE expressions. For example:

```
SELECT name,
       CASE
         WHEN jobcode IS NOT NULL THEN job_of2 (jobcode)
         ELSE NULL
       END
FROM emps
WHERE CASE
        WHEN jobcode IS NOT NULL THEN job_of2 (jobcode)
        ELSE NULL
      END<> 'Administrator';
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

You can also make such CASE expressions implicit, by specifying the RETURNS NULL ON NULL INPUT option in the CREATE FUNCTION statement:

```
CREATE FUNCTION job_of22 (jc INTEGER) RETURNS CHARACTER VARYING(20)
  RETURNS NULL ON NULL INPUT
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines5_jar:Routines5.job2';
```

When an SQL function is called whose CREATE FUNCTION statement specifies RETURNS NULL ON NULL INPUT, then if the runtime value of any argument is null, the result of the function call is set to null, and the function itself is not invoked.

The following SELECT statement invokes the `job_of22` function.

```
SELECT name, job_of22(jobcode)
FROM emps
WHERE job_of22(jobcode) <> 'Administrator';
```

This SELECT is equivalent to the previous SELECT that invokes the `job_of2` function within CASE expressions. That is, the RETURNS NULL ON NULL INPUT clause in the CREATE FUNCTION statement for `job_of22` makes the null-testing CASE expressions implicit.

The RETURNS NULL ON NULL INPUT option applies to *all* of the parameters of the function, not just to the parameters whose Java data type is not nullable.

The convention that the RETURNS NULL ON NULL INPUT option defines for a function is the same convention that is followed for most built-in SQL functions and operators: if any operand is null, then the value of the operation is null.

The alternative to the RETURNS NULL ON NULL INPUT clause is CALLED ON NULL INPUT, which is the default.

You can specify the same Java method in multiple CREATE FUNCTION statements (*i.e.*, defining SQL synonyms), and those CREATE FUNCTION statements can either specify RETURNS NULL ON NULL INPUT or CALLED ON NULL INPUT, as illustrated by the above `job_of2` and `job_of22`.

If you create multiple SQL functions named `job_of22` (with different numbers and/or types of parameters), you can specify (or default to) CALLED ON NULL INPUT in some of the CREATE FUNCTION `job_of22` statements, and specify RETURNS NULL ON NULL INPUT in others. The actions of the RETURNS NULL ON NULL INPUT clause are taken after overloading resolution has been done and a particular CREATE FUNCTION statement has been identified.

The RETURNS NULL ON NULL INPUT and CALLED ON NULL INPUT clauses can only be specified in CREATE FUNCTION statements, that is, not in CREATE PROCEDURE statements. This is because there is no equivalent conditional treatment of procedure calls that would be as generally useful.

H.16 Static variables

Java static methods can be contained in Java classes that have static variables, and, in Java, static methods can both reference and set static variables. For example:

```
public class Routines6 {
  static String jobs;
```

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  public static void setJobs (String js) throws SQLException {jobs=js;}
  public static String job3(int jc) throws SQLException {
    if (jc < 1 || jc * 5 > length(jobs)+1) return "Invalid jobcode";
    else return jobs.substring(5*(jc-1), 5*jc);
  }
}
```

Assume that you install this class in an SQL system:

```
SQLJ.INSTALL_JAR('file:~/classes/Routines6.jar', 'routines6_jar', 0);
```

The class `Routines6` has a static variable `jobs`, which is set by the static method `setJobs` and referenced by the static method `job3`. A class such as `Routines6` that dynamically modifies the values of static variables is well-defined in Java, and can be useful. However, when such a class is installed in an SQL system, and the methods `setJobs` and `job3` are defined as SQL procedures and functions (<SQL-invoked routine>), the scope of the assignments to the static variable `jobs` is implementation-dependent. That is, the scope of that variable is not specified, and is likely to differ across implementations (and possibly across the releases of a given implementation).

For example:

```
CREATE PROCEDURE set_jobs (js CHARACTER VARYING(100))
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines6_jar:Routines6.setJobs';
CREATE FUNCTION job_of3 (jc integer) RETURNS CHARACTER VARYING(20)
  RETURNS NULL ON NULL INPUT
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'routines6_jar:Routines6.job3';
CALL set_jobs ('AdminSalesClerk');
SELECT name, job_of3 (jobcode)
FROM emps
WHERE job_of3(jobcode) <> 'Admin';
```

This appears to be a straightforward use of the `Routines6` class in SQL. The call of `set_jobs` specifies a list of job code values, which a user might reasonably assume is "cached" by the SQL-environment and used in subsequent calls of `job_of3`. However, since the scope of the static variable `jobs` in the SQL-environment is implementation-dependent, the answers to the following questions regarding the values passed to the `set_jobs` procedure are likely to differ across implementations:

— Is the `set_jobs` value visible only to the current session? Or also to concurrent sessions and to later non-concurrent sessions?

— Does the `set_jobs` value persist across a COMMIT? Is it reset by a ROLLBACK?

The implication of this uncertainty is that you should not use classes that assign to static variables in SQL. Note, however, that such assignments will not (necessarily) be detected by the SQL implementation, either when you CREATE PROCEDURE/FUNCTION or when you call a routine.

You can prevent assignments to static variables in Java by declaring them with the `final` property.

H.17 Dropping SQL names of Java methods

After you have created SQL procedure or function names for Java methods, you can drop those SQL names with a normal SQL DROP statement:

```
DROP FUNCTION region RESTRICT;
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

A DROP statement has no effect on the Java method (or class) on which the SQL name was defined. Dropping an SQL procedure or function implicitly revokes any granted privileges for that routine.

H.18 Removing Java classes from SQL

You can completely uninstall a JAR with the SQLJ.REMOVE_JAR procedure. For example:

```
SQLJ.REMOVE_JAR ('routines_jar', 0);
```

As noted earlier, JARs can contain *deployment descriptors*, which specify implicit actions to be taken by the SQLJ.INSTALL_JAR and SQLJ.REMOVE_JAR procedures. The second parameter is an integer that specifies whether you do or do not (indicated by non-zero or zero values, respectively) want the SQLJ.REMOVE_JAR procedure to execute the actions specified by a deployment descriptor in the JAR. Deployment descriptors are further described in Subclause 12.2, "Deployment descriptor files".

After the SQLJ.REMOVE_JAR procedure performs any actions specified by the JAR's deployment descriptor file(s), there shall be no remaining SQL procedure or function whose external name references any method of any class in the specified JAR. Any such remaining SQL procedures or functions shall be explicitly dropped before the SQLJ.REMOVE_JAR procedure will be able to complete successfully.

H.19 Replacing Java classes in SQL

Assume that you have installed a Java JAR in SQL, and you want to replace some or all of the contained classes, *e.g.*, to correct or improve them. You can do this by using the SQLJ.REMOVE_JAR procedure to remove the current JAR, and then using the SQLJ.INSTALL_JAR procedure to install the new version. However, you will probably have executed one or more SQL DDL statements that depend on the methods of the classes that you want to replace. That is, you may have executed one or more of the following DDL operations:

— CREATE PROCEDURE/FUNCTION statements referencing the classes.

— GRANT statements referencing those SQL procedures and functions.

— CREATE PROCEDURE/FUNCTION statements for SQL procedures and functions that invoke those SQL procedures and functions.

— CREATE VIEW/TABLE statements for SQL views and tables that invoke those SQL procedures and functions.

The rules for the SQLJ.REMOVE_JAR procedure require that you drop all SQL procedure/functions that directly reference methods of a class before you can remove the JAR containing the class. And, SQL rules for RESTRICT, as specified in the SQL <drop routine statement>, require that you drop all SQL objects (tables, views, SQL-server modules, and routines whose bodies are written in SQL) that invoke a procedure/function before you drop the procedure/function.

Thus, if you use the SQLJ.REMOVE_JAR and SQLJ.INSTALL_JAR procedures to replace a JAR, you will have to drop the SQL objects that directly or indirectly depend on the methods of the classes in the JAR, and then re-create those items.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The SQLJ.REPLACE_JAR procedure avoids this requirement, by performing an instantaneous *remove* and *install*, with suitable validity checks. You can therefore call the SQLJ.REPLACE_JAR procedure without first dropping the dependent SQL objects.

For example, in Subclause H.4, "Installing region and correctStates in SQL", we installed the class of Routines1 with the following statement:

```
SQLJ.INSTALL_JAR( 'file:~/classes/Routines1.jar', 'routines1_jar', 0)
```

You can replace that JAR with a statement such as:

```
SQLJ.REPLACE_JAR( 'file:~/revised_classes/Routines1.jar', 'routines1_jar')
```

Note that the JAR name shall be the same. It identifies the existing JAR, and will subsequently identify the replacement JAR. The URL of the replacement JAR can be the same as or different from the URL of the original JAR.

In the general case, there will be classes in the old JAR that are not in the new JAR, classes that are in both JARs, and classes that are in the new JAR and not in the old JAR. These are referred to respectively as unmatched old classes, matching old/new classes, and unmatched new classes.

The validity requirements on the replacement JAR are:

— There shall be no SQL procedure or function whose routine descriptor's <external routine name> specified an <external Java reference string> that references any method of any unmatched old class (since all unmatched old classes will be removed).

— Any CREATE PROCEDURE/FUNCTION statement that references a method of a matching class shall be a valid statement for the new class.

— There shall be no SQL user-defined type whose descriptor's <jar and class name> references any unmatched old class.

— Any CREATE TYPE statement that references a method of a matching class shall be a valid statement for the new class.

If these requirements are satisfied, the SQLJ.REPLACE_JAR procedure deletes the old classes (both unmatched and matching) and installs the new classes (both unmatched and matching).

H.20 Visibility

The SQLJ.INSTALL_JAR procedure will install any Java classes into the SQL system. However, not all methods of all classes can be referenced in SQL. Only *visible* methods of *visible* classes can be referenced in SQL. The notion of visible classes and methods is based on the concept of *mappable* data types. The detailed definitions of *mappable* and *visible* are specified in Subclause 4.5, "Parameter mapping". They may be summarized as follows:

— A Java data type is *mappable* to SQL (and vice versa) if and only if it has a corresponding SQL data type, or it is an array that is used for OUT parameters, or it is an array that is used for result sets.

— A Java method is *mappable* (to SQL) if and only if the data type of each parameter is mappable, and the result type is either a mappable data type or is void.

A Java method is *visible* in SQL if and only if it is public, static, and mappable.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Only the visible installed methods can be referenced in SQL. Other methods simply don't exist in SQL. Attempts to reference them will raise implementation-defined syntax errors such as *unknown name*.

Non-visible classes and methods can, however, be used by the visible methods.

H.21 Exceptions

SQL exception conditions are defined for the SQL/JRT procedures. For example, if the URL argument specified in calls to `SQLJ.INSTALL_JAR` or `SQLJ.REPLACE_JAR` (*etc*.) is invalid, an SQL exception condition (`java.sql.SQLException`) with a specified SQLSTATE will be raised. These exception conditions are specified in the definitions of the procedures, and are listed in Subclause 15.1, "SQLSTATE". Java exceptions that are thrown during execution of a Java method in SQL can be caught within Java, and if this is done, then those exceptions do not affect SQL processing.

Any Java exceptions that are uncaught when a Java method called from SQL completes will be returned in SQL as SQL exception conditions.

For example, in Subclause H.3, "Example Java methods: region and correctStates", we defined the Java method `Routines1.region`. And, in Subclause H.5, "Defining SQL names for region and correctStates", we defined the SQL function name `region_of` for the Java method `Routines1.region`.

The Java method `Routines1.region` throws an exception if the argument value is not in a specified range of values:

```
public class routines1 {
  public static int region(String s) throws SQLException {
    if (s.equals ("MN") || s.equals ("VT") || s.equals ("NH")) return 1;
    else if (s.equals ("FL") || s.equals ("GA") || s.equals ("AL")) return 2;
    else if (s.equals ("CA") || s.equals ("AZ") || s.equals ("NV")) return 3;
    else throw new SQLException("Invalid state code", "38001");
  }
}
```

Assume that the EMPS table contains a row for which the value of the STATE column is 'TX'. The following SELECT will therefore raise an exception condition when it encounters that row of EMPS:

```
SELECT name, region_of(state)
FROM emps
WHERE region_of(state) = 1;
```

The call of the `region_of` function with an invalid parameter ('TX') will raise the SQL exception condition with the SQLSTATE of '38001'. The SQL message text associated with that exception will be the following string:

```
'Invalid state code'
```

The message text and SQLSTATE may be specified in the Java exception specified in the Java `throw` statement. If that exception does not specify an SQLSTATE, 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 `SQLException` subclass of the Java `Exception` class. The effect of such an SQL exception condition on the outer SQL statement that called the

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Java method is implementation-defined. For portability, a Java method that is called from SQL, that itself executes an SQL statement, and that catches an `SQLException` from that inner SQL statement should re-throw that `SQLException`.

H.22 Deployment descriptors

When you install a JAR containing a set of Java classes into SQL, you shall execute one or more CREATE PROCEDURE/FUNCTION statements before you can call the static methods of those classes as SQL procedures and functions. And, you may also want to perform various GRANT statements for the SQL names created by those CREATE PROCEDURE FUNCTION statements. When you later remove a JAR, you will want to execute corresponding DROP PROCEDURE/FUNCTION statements and REVOKE statements.

If you plan to install a JAR in several SQL systems, the various CREATE, GRANT, DROP, and REVOKE statements will often be the same for each such SQL system. One way that you could simplify the install and remove actions would be as follows:

— Provide methods called "`afterInstall`" and "`beforeRemove`" to be executed as an "install script" and "remove script", performing such actions as the following:

  • The `afterInstall` method: The CREATE and GRANT statements that you want to be performed when the JAR is installed.

  • The `beforeRemove` method: The DROP and REVOKE statements (the inverse of the actions of the `afterInstall` method) that you want to be performed when the JAR is removed.

  That is, the `afterInstall` and `beforeRemove` methods would use SQL/OLB or JDBC to invoke SQL for the desired CREATE, GRANT, DROP, and REVOKE statements.

— Include the `afterInstall` and `beforeRemove` methods in a class, which you might call the `deploy` class, and include that `deploy` class in the JAR that you plan to distribute.

— Instruct recipients of the JAR to do the following to install the JAR:

  • Call the `SQLJ.INSTALL_JAR` procedure for the JAR.

  • Execute a CREATE procedure statement for the `afterInstall` method, giving it an SQL name such as `after_install`. Note that this "bootstrap" action cannot be included in the `afterInstall` method itself.

  • Call the `after_install` procedure. Note: We can assume that the `after_install` procedure will include a CREATE PROCEDURE statement to give the `beforeRemove` method an SQL name such as `before_remove`.

— Instruct recipients of the JAR to proceed as follows to remove the JAR:

  • Call the `before_remove` procedure.

  • Drop the `after_install` and `before_remove` procedures. Note that this action cannot be included in the `beforeRemove` procedure itself.

  • Call the `SQLJ.REMOVE_JAR` procedure.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Note that this simplification of the install and remove actions still requires several manual steps. SQL/JRT therefore provides a mechanism, called *deployment descriptors*, with which you can specify the SQL statements that you want to be executed implicitly by the SQLJ.INSTALL_JAR and SQLJ.REMOVE_JAR procedures.

If you want the deployment descriptors in a JAR to be interpreted when you install and remove the JAR, then you specify a non-zero value for the deploy parameter of the SQLJ.INSTALL_JAR procedure and similarly for the undeploy parameter of the SQLJ.REMOVE_JAR procedure. If a JAR contains a deployment descriptor, then the SQLJ.INSTALL_JAR procedure will use that deployment descriptor to determine the CREATE and GRANT statements to execute after it has installed the classes of the JAR. The corresponding SQLJ.REMOVE_JAR procedure uses the deployment descriptor to determine the DROP and REVOKE statements to execute before it removes the JAR and its classes.

A deployment descriptor is a text file containing a list of SQL CREATE and GRANT statements to be executed when the JAR is installed, and a list of SQL DROP and REVOKE statements to be executed when the JAR is removed.

For example, suppose that you have incorporated the above classes Routines1, Routines2, and Routines3 into a single JAR. The following is a possible deployment descriptor that you might want to include in that JAR.

Notes:

— Within a deployment descriptor file, you use the JAR name "thisjar" as a placeholder JAR name in the EXTERNAL NAME clauses of CREATE statements.

— The various user names in this example are of course hypothetical.

```
SQLActions[ ] = {
  "BEGIN INSTALL
    CREATE PROCEDURE correct_states (old CHARACTER(20), new CHARACTER(20))
      MODIFIES SQL DATA
      LANGUAGE JAVA PARAMETER STYLE JAVA
      EXTERNAL NAME 'thisjar:Routines1.correctStates';
    GRANT EXECUTE ON correct_states TO Baker;
    CREATE FUNCTION region_of(state CHARACTER(20)) RETURNS INTEGER
      NO SQL
      LANGUAGE JAVA PARAMETER STYLE JAVA
      EXTERNAL NAME 'thisjar:Routines1.region';
    GRANT EXECUTE ON region_of TO PUBLIC;
    CREATE PROCEDURE best2 (OUT n1 CHARACTER VARYING(50), OUT id1 CHARACTER(5),
                            OUT region1 INTEGER, OUT s1 DECIMAL(6,2),
                            OUT n2 CHARACTER VARYING(50), OUT id2 CHARACTER(5),
                            OUT region2 INTEGER, OUT s2 DECIMAL(6,2),
                            region INTEGER)
      READS SQL DATA
      LANGUAGE JAVA PARAMETER STYLE JAVA
      EXTERNAL NAME 'thisjar:Routines2.bestTwoEmps';
    GRANT EXECUTE ON best2 TO Baker, Cook, Farmer;
    CREATE PROCEDURE ordered_emps (region INTEGER)
      READS SQL DATA
      DYNAMIC RESULT SETS 1
      LANGUAGE JAVA PARAMETER STYLE JAVA
      EXTERNAL NAME 'thisjar:Routines3.rankedEmps';
    GRANT EXECUTE ON ordered_emps TO PUBLIC;
  END INSTALL",
  "BEGIN REMOVE
```

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    REVOKE EXECUTE ON correct_states FROM Baker RESTRICT;
    DROP PROCEDURE correct_states RESTRICT;
    REVOKE EXECUTE ON region_of FROM PUBLIC RESTRICT;
    DROP FUNCTION region_of RESTRICT;
    REVOKE EXECUTE ON best2 FROM Baker, Cook, Farmer RESTRICT;
    DROP PROCEDURE best2 RESTRICT;
    REVOKE EXECUTE ON ordered_emps FROM PUBLIC RESTRICT;
    DROP PROCEDURE ordered_emps RESTRICT;
  END REMOVE"
}
```

Assume that `deploy_routines.txt` is the name of a text file containing the above deployment descriptor. You would build a JAR containing the following:

— The text file `deploy_routines.txt`.

— The class files for `Routines1`, `Routines2`, and `Routines3`.

— A manifest file with the following manifest entry:

```
Name: deploy_routines.txt
SQLJDeploymentDescriptor: TRUE
```

This manifest entry identifies the file `deploy_routines.txt` as a deployment descriptor in the JAR, for the `SQLJ.INSTALL_JAR` and `SQLJ.REMOVE_JAR` procedures to interpret.

Deployment descriptor files can contain syntax errors. In general, any error that can arise in a CREATE or GRANT statement can occur in a deployment descriptor file.

You may want to install a JAR that contains a deployment descriptor file without performing the deployment actions. For example, those actions may contain syntax errors, or may simply be inappropriate for some SQL system. You can do this by specifying a zero value for the `deploy` parameter of the `SQLJ.INSTALL_JAR` procedure, and a zero value for the `undeploy` parameter of the `SQLJ.REMOVE_JAR` procedure.

H.23 Paths

In the preceding clauses, the example JARs and their Java classes referenced other Java classes in the packages `java.lang` and `java.sql`. The JARs and their Java classes that you install can also reference Java classes in other JARs that you have installed or will install. For example, suppose that you have three JARs, containing Java classes relating to administration, project management, and property management.

```
SQLJ.INSTALL_JAR ('file:~/classes/admin.jar', 'admin_jar', 0);
```

At this point, you can execute CREATE PROCEDURE/FUNCTION statements referencing the methods of classes in `admin_jar`. And, you can call those procedures and functions. If, at runtime, the Java methods of `admin_jar` reference system classes or other Java classes that are contained in `admin_jar`, then those references will be resolved implicitly. If the `admin_jar` methods reference Java classes that are contained in `property_jar` (which we will install below), then an exception condition will be raised for an unresolved class reference.

```
SQLJ.INSTALL_JAR ('file:~/classes/property.jar', 'property_jar', 0);
SQLJ.INSTALL_JAR ('file:~/classes/project.jar', 'project_jar', 0);
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

These calls of SQLJ.INSTALL_JAR install property_jar and project_jar. However, references to the property_jar classes by classes in admin_jar will still not be resolved. Similarly, references within property_jar to classes in project_jar will not be resolved, and vice versa.

To summarize:

— When you install a JAR, any references within the classes of that JAR to system classes, or to classes that are contained in the same JAR, will be implicitly resolved.

— References to any other classes, installed or not, are unresolved.

— You can install JARs that have unresolved class references, and you can use CREATE PROCEDURE/FUNCTION statements to define SQL routines on the methods of those classes.

— When you call SQL routines defined on Java methods, exceptions for unresolved class references may occur at any time allowed by [Java].

Invoking classes that contain unresolved references can be useful:

— To use or to test partially-written applications.

— To use classes that have some methods that are not appropriate for use in an SQL-environment. For example, a class that has display-oriented or interactive methods that are used in other Java-enabled environments, but not within an SQL system.

To resolve references to classes in other JARs, you use the SQLJ.ALTER_JAVA_PATH procedure.

```
SQLJ.ALTER_JAVA_PATH ('admin_jar', '(property.*,property_jar)
                                    (project.*, project_jar)');
SQLJ.ALTER_JAVA_PATH ('property_jar', '(project.*,project_jar)');
SQLJ.ALTER_JAVA_PATH ('project_jar', '(*, property_jar) (*, admin_jar)');
```

The SQLJ.ALTER_JAVA_PATH procedure has two arguments, both of which are character strings. In a call SQLJ.ALTER_JAVA_PATH(JX, PX):

— JX is the name of the JAR for which you want to specify a path. This is the JAR name that you specified in the INSTALL_JAR procedure.

— PX is the path of JARs in which you want unresolved class names that are referenced by classes contained in JX to be resolved. The path argument is a character string containing a list of path elements (not comma-separated). Each path element is a parenthesized pair (comma-separated), in which the first item is a pattern, and the second item is a JAR name.

Suppose that at runtime, some method of a class C that is contained in JAR JX is being evaluated. And, suppose that within the execution of class C, a reference to some other class named XC is encountered, such that no class named XC is defined in JAR JX. The path PX specified for JAR JX in the SQLJ.ALTER_JAVA_PATH call determines the resolution, if any, of class name XC:

— Each path element '(PAT_i, J_i)' is examined.

— If PAT_i is a fully qualified class name that is equivalent to XC, then XC shall be defined in JAR J_i. If it is not, then the reference to XC is unresolved.

— If PAT_i is a package name followed by an '*', and XC is the name of a class in that package, then XC shall be defined in JAR J_i. If it is not, then the reference to XC is unresolved.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— If PAT_i is a single '*', then if XC is defined in JAR J_i, that resolution is used; otherwise, subsequent path elements are tested.

The paths that we specified above for `admin_jar`, `property_jar`, and `project_jar` therefore have the following effect:

— When executing within `admin_jar`, classes that are in the `property` or `project` packages will be resolved in `property_jar` and `project_jar`, respectively.

— When executing within `property_jar`, classes that are in the `project` package will be resolved in `project_jar`.

— When executing within `project_jar`, all classes will first be resolved in `property_jar`, and then in `admin_jar`.

Note that if a class C contained in `property_jar` directly contains a reference to a class AC contained in `admin_jar`, then that reference to AC will be unresolved, since `admin_jar` is not specified in the path for `property_jar`. But, if that class C invokes a method `project.C2.M` of a class contained in `project_jar`, and `project.C2.M` references class AC, then that reference to AC will be resolved in `admin_jar`, since `admin_jar` is specified in the path for `project_jar`. That is, while class C is being executed, the path specified for `property_jar` is used, and while class C2 is being executed, the path specified for `project_jar` is used. Thus, as execution transfers to classes contained in different JARs, the current path changes to the path specified for each such JAR. In other words, the path specified for a JAR J1 applies only to class references that occur directly within the classes of J1, not to class references that occur in some class contained in another JAR that is invoked from a class of J1.

The path that you specify in a call of the `SQLJ.ALTER_JAVA_PATH` procedure becomes a property of the specified JAR. A given JAR has at most one path. The path (if any) for a JAR applies to all users of the classes and methods in the JAR.

When you call the `SQLJ.ALTER_JAVA_PATH` procedure, the path you specify replaces the current path (if any) for the specified JAR. The effect of this replacement on currently running classes and methods is implementation-defined.

When you execute the `SQLJ.ALTER_JAVA_PATH` procedure, you shall be the owner of the JAR that you specify as the first argument, and you shall have the USAGE privilege on each JAR that you specify in the path argument.

The path facility is an optional feature.

H.24 Privileges

The SQL privilege system is extended for SQL/JRT.

First, the SQLJ build-in procedures are considered to be SQL-schema statements, and as such require implementation-defined privileges to be invoked.

Second, the USAGE privilege is defined for JARs. USAGE is needed on a JAR in order to:

— Reference it in a CREATE PROCEDURE/FUNCTION/TYPE statement.

— List it in an SQL-Java path in an `SQLJ.ALTER_JAVA_PATH` procedure call.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The user who installs a JAR is the owner of that JAR and implicitly has USAGE on the JAR, and can grant USAGE to other users and roles. Only the owner can replace, remove, or alter the JAR.

USAGE privileges on a JAR is an optional feature.

H.25 Information Schema

Additional views and columns are defined for the Information Schema to describe external Java routines and external Java types:

— JARS lists the JARs installed in a database.

— METHOD_SPECIFICATIONS is augmented to include information about static field methods.

— ROUTINES contains information about external Java routines.

— USAGE_PRIVILEGES contains information on USAGE privileges granted on JARs.

— USER_DEFINED_TYPES is augmented to include information about external Java types.

In addition, the usage of JARs by routines, types, and other JARs is shown in a collection of new usage views:

— JAR_JAR_USAGE lists the JARs used in the SQL-Java path of a given JAR.

— ROUTINE_JAR_USAGE names the JAR used in an external Java routine.

— TYPE_JAR_USAGE names the JAR used in an external Java type.

These Information Schema views are optional features.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex I
(informative)

Types tutorial

I.1 Overview

This tutorial clause shows a series of example Java classes and their methods, and shows how they can be installed in an SQL system and used as data types in SQL.

I.2 Example Java classes

This Subclause shows example Java classes `Address` and `Address2Line`.

— The `Address` class represents street addresses in the USA, with a `street` field containing a street name and building number, and a `zip` field containing a postal code.

— The `Address2Line` class is a subclass of the `Address` class. It adds one additional field, named `line2`, which would contain data such as an apartment number.

— The `Address` and `Address2Line` classes both have the following methods:

  • A default niladic constructor.

  • A constructor with parameters.

  • A `toString` method to return a string representation of an address.

— The `Address` and `Address2Line` classes are both specified to implement the Java interfaces `java.io.Serializable` and `java.sql.SQLData`.

A Java class that will be used as a data type in SQL shall implement either the Java interface `java.io.Serializable` or the Java interface `java.sql.SQLData` or both. This is required to transfer class instances between JVMs and between Java and SQL.

It is assumed that the import statements `import java.sql.*;` and `java.math.*;` have been included in all classes.

The following is the text of the `Address` class:

```
public class Address implements java.io.Serializable, java.sql.SQLData {
    public String street;
    public String zip;
    public static int recommendedWidth = 25;
    private String sql_type; // For the java.sql.SQLData interface
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    // A default constructor
    public Address ( ) {
      street = "Unknown";
      zip = "None";
    }
    // A constructor with parameters
    public Address (String S, String Z) {
      street = S;
      zip = Z;
    }
    // A method to return a string representation of the full address
    public String toString( ) {
    return "Street=" + street + " ZIP=" + zip;
    }
    // A void method to remove leading blanks
    // This uses the static method Misc.stripLeadingBlanks.
    public void removeLeadingBlanks( ) {
      street = Misc.stripLeadingBlanks(street);
      zip = Misc.stripLeadingBlanks(zip);
    }
    // A static method to determine if two addresses
    // are in arithmetically contiguous zones.
    public static String contiguous(Address a1, Address a2) {
      if (Integer.parseInt(a1.zip) == Integer.parseInt(a2.zip)+1 ||
          Integer.parseInt(a1.zip) == Integer.parseInt(a2.zip) -1)
        return("yes");
      else
        return("no");
    }
    // java.sql.SQLData implementation:
    public void readSQL (SQLInput in, String type)
        throws SQLException {
      sql_type = type;
      street = in.readString();
      zip = in.readString();
    }
    public void writeSQL (SQLOutput out)
        throws SQLException {
      out.writeString(street);
      out.writeString(zip);
    }
    public String getSQLTypeName () {
      return sql_type;
    }
}
```

The following is the text of the `Address2Line` class, which is a subclass of the `Address` class:

```
public class Address2Line extends Address
        implements java.io.Serializable, java.sql.SQLData {
    public String line2;
    // A default constructor
    public Address2Line () {
      super() ;
      line2 = " ";
    }
    // A constructor with parameters
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    public Address2Line (String S, String L2, String Z) {
      street = S;
      line2 = L2;
      zip = Z;
  }
    // A method to return a string representation of the full address
    public String toString() {
      return "Street=" + street +" Line2=" + line2 + " ZIP=" + zip;
    }
    // A void method to remove leading blanks.
    // Note that this is an imperative method that modifies the instance.
    // This uses the static method Misc.stripLeadingBlanks defined below.
    public void removeLeadingBlanks( ) {
      line2 = Misc.stripLeadingBlanks(line2);
      super.removeLeadingBlanks() ;
    }
    // java.sql.SQLData implementation:
    public void readSQL (SQLInput in, String type)
        throws SQLException {
      super.readSQL(in,type);
      line2 = in.readString();
    }
    public void writeSQL (SQLOutput out)
        throws SQLException {
      super.writeSQL(out);
      out.writeString(line2);
    }
}
    //The following class and method is used only internally in the above Java methods.
    //We won't define an SQL function for this method.
public class Misc {
    // remove leading blanks from a String
    public static String stripLeadingBlanks(String s) {
      int scan;
      for (scan=0; scan < s.length() ; scan++)
        if ( !java.lang.Character.isSpace(s.charAt(scan)) )
          break;
      if (scan == s.length() ) return"";
      else return s.substring(scan);
    }
}
```

I.3 Installing Address and Address2Line in an SQL system

To install classes such as `Address` and `Address2Line` in an SQL system, you proceed as in Annex H, "Routines tutorial". The source code for the classes will be in files with filetype `java`, which you compile using the `javac` command to produce object code files with filetype `class`. You then assemble those `class` files into a Java JAR with filetype `jar`, and you place that JAR in a directory for which you can specify a URL. Assume that `file:~/classes/AddrJar.jar` is such a URL. Now, you can install the classes into an SQL system by calling the `SQLJ.INSTALL_JAR` procedure that was described in Annex H, "Routines tutorial":

```
SQLJ.INSTALL_JAR ('file:~/classes/AddrJar.jar', 'address_classes_jar', 0);
```

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

I.4 CREATE TYPE for Address and Address2Line

Before you can use a Java class as an SQL data type, you shall define SQL names for the SQL data type and its fields and methods. You do this with extended forms of the SQL CREATE TYPE statement.

An implementation of this part of ISO/IEC 9075 may support these extended forms of the CREATE TYPE statement explicitly as standalone SQL statements, or in deployment descriptor files, or may support an implementation-defined mechanism that achieves the same effect as the CREATE TYPE statement. Deployment descriptor files are included in JARs, and executed implicitly during calls of the built-in SQL/JRT procedure `SQLJ.INSTALL_JAR` that specify a deploy action (third parameter non-zero). This is described in Subclause H.22, "Deployment descriptors". In this Annex, we will show the CREATE TYPE statements as standalone SQL statements.

The following SQL CREATE TYPE statements reference the above Java `Address` and `Address2Line` classes:

```
CREATE TYPE addr EXTERNAL NAME 'address_classes_jar:Address'
     LANGUAGE JAVA
      AS (
    street_attr       CHARACTER VARYING(50) EXTERNAL NAME 'street',
    zip_attr          CHARACTER(10) EXTERNAL NAME 'zip' )
    STATIC METHOD rec_width ()
      RETURNS INTEGER
      EXTERNAL VARIABLE NAME 'recommendedWidth',
    CONSTRUCTOR METHOD addr ()
      RETURNS addr SELF AS RESULT
      EXTERNAL NAME 'Address',
    CONSTRUCTOR METHOD addr (s_parm CHARACTER VARYING(50),
                             z_parm CHARACTER(10))
      RETURNS addr SELF AS RESULT
      EXTERNAL NAME 'Address',
    METHOD to_string ()
      RETURNS CHARACTER VARYING(255)
      EXTERNAL NAME 'toString',
    METHOD remove_leading_blanks ()
      RETURNS addr SELF AS RESULT
      EXTERNAL NAME 'removeLeadingBlanks',
    STATIC METHOD contiguous (A1 addr, A2 addr)
      RETURNS CHARACTER(3)
      EXTERNAL NAME 'contiguous';
CREATE TYPE addr_2_line
      UNDER addr
      EXTERNAL NAME 'address_classes_jar:Address2Line'
      LANGUAGE JAVA
      AS (
    line2_attr        CHARACTER VARYING (100) EXTERNAL NAME 'line2' )
    CONSTRUCTOR METHOD addr_2_line ()
      RETURNS addr_2_line SELF AS RESULT
      EXTERNAL NAME 'Address2Line',
    CONSTRUCTOR METHOD addr_2_line (s_parm  CHARACTER VARYING(50),
                                    s2_parm CHARACTER(100),
                                    z_parm  CHARACTER(10))
      RETURNS addr_2_line SELF AS RESULT
      EXTERNAL NAME 'Address2Line',
    METHOD strip ()
```

 © ISO/IEC 2008 – All rights reserved

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      RETURNS addr_2_line SELF AS RESULT
      EXTERNAL NAME 'removeLeadingBlanks';
```

These CREATE TYPE statements are an extension of the SQL CREATE TYPE statement. The above extensions add the EXTERNAL clauses, which are patterned after the EXTERNAL clause of the SQL CREATE PROCEDURE/FUNCTION statement, and the METHOD clauses, which are patterned after SQL CREATE PROCEDURE/FUNCTION statements.

In this Annex, we'll describe the basic elements of these CREATE TYPE statements, and defer to later sections discussions of the following less intuitive clauses:

— The Java static field `recommendedWidth` of the `Address` class is represented in the SQL CREATE TYPE by a static method with no arguments, named `rec_width`. This is described in Subclause I.15, "Static fields".

— The Java `void` method `removeLeadingBlanks` of the `Address` class is represented in the SQL CREATE TYPE for the `addr` type by a method, `remove_leading_blanks` that specifies RETURNS SELF AS RESULT. The `removeLeadingBlanks` and `strip` methods of the `Address2Line` class are treated similarly. This is described in Subclause I.16, "Instance-update methods". The `strip` method is included to illustrate that multiple SQL methods can reference a single Java method.

— The other clauses of the CREATE TYPE statements are straightforward transliterations of the signatures of the Java classes.

The EXTERNAL clause following the CREATE TYPE clause shall reference a Java class that is in its identified installed JAR. This is referred to as the *subject Java class*, and the SQL data type is the *subject SQL data type*.

If the EXTERNAL clause of a METHOD clause references a Java constructor method (*i.e.*, a method with no explicitly specified return type whose name is the same as the class name), then the SQL method name shall be the same as the SQL data type name. That is, the same conventions for constructor function calls will be used in SQL as in Java.

SQL data types such as `addr` and `addr_2_line` that are defined on Java classes are referred to as *external Java data types*.

I.5 Multiple SQL types for a single Java class

You can define more than one SQL data type on a given Java class. For example:

```
CREATE TYPE another_addr
      EXTERNAL NAME 'address_classes_jar:Address'
      LANGUAGE JAVA
      AS (
    zip_part    CHARACTER(10) EXTERNAL NAME 'zip',
    street_part CHARACTER VARYING(50) EXTERNAL NAME 'street')
    STATIC METHOD rec_width_part () RETURNS INTEGER
      EXTERNAL VARIABLE NAME 'recommendedWidth',
    CONSTRUCTOR METHOD another_addr ()
      RETURNS another_addr SELF AS RESULT
      EXTERNAL NAME 'Address',
    CONSTRUCTOR METHOD another_addr (s_parm CHARACTER VARYING(50),
                                     z_parm CHARACTER(10))
      RETURNS another_addr SELF AS RESULT
```

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      EXTERNAL NAME 'Address',
    METHOD string_rep ()
      RETURNS CHARACTER VARYING(255)
      EXTERNAL NAME 'toString',
    STATIC METHOD contig (A1 another_addr,
                          A2 another_addr)
      RETURNS CHARACTER(3)
      EXTERNAL NAME 'contiguous';
```

The SQL data type `another_addr` is a different data type than the `addr` data type. The two data types aren't comparable, assignable, or union compatible. You can include or omit an SQL data type that is a subtype of the `another_addr` type for "2 line" data. If you define such a subtype, with a name such as `another_2_line`, then instances of `another_2_line` are specializations of `another_addr`, and not of `addr`.

I.6 Collapsing subclasses

Given Java classes and subclasses such as `Address` and `Address2Line`, you can either define SQL data types for each such class, or for a subset of those classes.

Assume that in SQL you only want to use the Java class `Address2Line`. You can define an SQL data type for that class without a corresponding SQL data type for the `Address` class. For example:

```
CREATE TYPE complete_addr
      EXTERNAL NAME 'address_classes_jar:Address2Line'
      LANGUAGE JAVA
      AS (
    zip_attr    CHARACTER(10) EXTERNAL NAME 'zip',
    street_attr CHARACTER VARYING(50) EXTERNAL NAME 'street',
    line2_attr  CHARACTER VARYING(100) EXTERNAL NAME 'line2' )
    STATIC METHOD rec_width ()
      RETURNS INTEGER
      EXTERNAL VARIABLE NAME 'recommendedWidth',
    CONSTRUCTOR METHOD complete_addr ()
      RETURNS complete_addr SELF AS RESULT
      EXTERNAL NAME 'Address2Line',
    CONSTRUCTOR METHOD complete_addr (s_parm  CHARACTER VARYING(50),
                                      s2_parm CHARACTER(100),
                                      z_parm  CHARACTER(10))
      RETURNS complete_addr SELF AS RESULT
      EXTERNAL NAME 'Address2Line',
    STATIC METHOD contiguous (A1 complete_addr,
                              A2 complete_addr)
      RETURNS CHARACTER(3)
      EXTERNAL NAME 'contiguous',
    METHOD to_string ()
      RETURNS CHARACTER VARYING(255)
      EXTERNAL NAME 'toString',
    METHOD strip ()
      RETURNS complete_addr SELF AS RESULT
      EXTERNAL NAME 'removeLeadingBlanks';
```

Note that this CREATE TYPE includes attribute and method definitions for attributes and methods of the superclass, `Addr`. You can include such superclass attributes and methods in a CREATE TYPE only if the

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

CREATE TYPE does not specify UNDER. That is, if a CREATE TYPE specifies a supertype with an UNDER clause, then the CREATE TYPE can only include attributes and methods of its immediate subject Java class.

The subsets of the classes that you can specify in CREATE TYPE statements are restricted. For example, assume that you install a hierarchy of classes `Person`, `Employee`, `Manager`, and `Director`, where each is a subclass of the preceding. You can then define SQL data types for the following subsets of the classes:

— `Person`, `Employee`, `Manager`, and `Director`: This is the full subset. Each SQL data type can include only members of its subject Java class.

— Any one of `Person`, `Employee`, `Manager`, or `Director`. That type can include members from any of its superclasses.

— `Manager` and `Director`: The SQL data type for `Manager` can include members from `Person` and `Employee`. The SQL data type for `Director` can include only members of `Director`.

— `Employee`, `Manager`, and `Director`: The SQL data type for `Employee` can include members from `Person`. The SQL data types for `Manager` and `Director` can include only members of those classes.

— `Employee` and `Manager`. The SQL data type for `Employee` can include members from `Person`. The SQL data types for `Manager` can include only members of that class.

— `Person`, `Employee`, and `Manager`, or `Person` and `Employee`. Each class can include only members of its subject Java class.

The subsets that are not allowed are those that omit an intermediate level of subclass. That is, you cannot define SQL data types for (only) the following subsets of the classes:

— `Person` and `Manager`, or `Person`, `Manager`, and `Director`.

— `Person` and `Director`.

— `Person`, `Employee`, and `Director`, or `Employee` and `Director`.

The rule is simpler than the explanation:

> If a CREATE TYPE statement for SQL type S2 specifies “UNDER S1”, then the subject Java class of S1 shall be the direct superclass of the subject Java class of S2.

Subclause I.5, “Multiple SQL types for a single Java class”, describes how you can define multiple SQL data types on a single Java class. This also can be done for subtype hierarchies. For example, let P_i, E_i, M_i, and D_i be SQL data types defined on `Person`, `Employee`, `Manager`, and `Director`. For a given number *i*, each type is defined to be a subtype of the preceding *i* type. You can define SQL data types such as:

— E1 and M1, and P2 and E2. That is, M1 is defined to be a subtype of E1, and E2 is defined to be a subtype of P2. In this case, E1 and E2 are different types. Instances of E1 are not specializations of P2.

— P1, E1, and M1, and M2 and D2. That is, E1 is defined to be a subtype of P1, M1 is defined to be a subtype of E1, and D2 is defined to be a subtype of M2. In this case, M1 and M2 are different types. Instances of M2 are not specializations of either P1 or E1, and instances of D2 are not specializations of either P1, E1, or M1.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

I.7 GRANT and REVOKE statements for data types

After you have performed the CREATE TYPE statements shown in the preceding clause, you can perform normal SQL GRANT statements to grant the SQL USAGE privilege on the new data type:

```
GRANT USAGE ON TYPE addr TO PUBLIC;

GRANT USAGE ON TYPE addr2line TO admin;
```

The syntax and semantics for GRANT and REVOKE of the USAGE privilege for user-defined types are as specified in [ISO9075-2], and are not further described by this part of ISO/IEC 9075.

I.8 Deployment descriptors for classes

You may want to perform the same set of SQL CREATE and GRANT statements in any SQL system in which you install a given JAR of Java classes, together with the corresponding SQL DROP and REVOKE statements when you remove that JAR. You can automate this process by specifying those SQL statements in a *deployment descriptor* file in the JAR. A deployment descriptor file contains a list of CREATE and GRANT statements to be executed when the JAR is installed, and a list of REVOKE and DROP statements to be executed when the JAR is removed.

The following is an example deployment descriptor file for the above Java classes and SQL CREATE and GRANT statements.

```
SQLActions[ ] = {
  "BEGIN INSTALL
     CREATE TYPE addr
         EXTERNAL NAME 'address_classes_jar:Address'
         LANGUAGE JAVA
         AS (
       zip_attr         CHARACTER(10) EXTERNAL NAME 'zip',
       street_attr      CHARACTER VARYING(50) EXTERNAL NAME 'street')
       STATIC METHOD rec_width()
         RETURNS INTEGER
         EXTERNAL VARIABLE NAME 'recommendedWidth',
       CONSTRUCTOR METHOD addr ()
         RETURNS addr SELF AS RESULT
         EXTERNAL NAME 'Address',
       CONSTRUCTOR METHOD addr (s_parm CHARACTER VARYING(50),
                                z_parm CHARACTER(10))
         RETURNS addr SELF AS RESULT
         EXTERNAL NAME 'Address',
       METHOD to_string ()
         RETURNS CHARACTER VARYING(255)
         EXTERNAL NAME 'toString',
       METHOD remove_leading_blanks ()
         RETURNS addr SELF AS RESULT
         EXTERNAL NAME 'removeLeadingBlanks',
       METHOD strip ()
         RETURNS addr SELF AS RESULT
         EXTERNAL NAME 'removeLeadingBlanks',
       STATIC METHOD contiguous (a1 addr, a2 addr)
         RETURNS CHARACTER(3)
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        EXTERNAL NAME 'contiguous';
    GRANT USAGE ON TYPE addr TO PUBLIC;
    CREATE TYPE addr_2_line UNDER addr
        EXTERNAL NAME 'address_classes_jar:Address2Line'
        LANGUAGE JAVA
        AS (
      line2_attr       CHARACTER VARYING(100) EXTERNAL NAME 'line2' )
      CONSTRUCTOR METHOD addr_2_line ()
        RETURNS addr_2_line SELF AS RESULT
       EXTERNAL NAME 'Address2Line',
      CONSTRUCTOR METHOD addr_2_line (s_parm  CHARACTER VARYING(50),
                                      s2_parm CHARACTER(100),
                                      z_parm CHARACTER(10) )
        RETURNS addr_2_line SELF AS RESULT
        EXTERNAL NAME 'Address2Line',
      METHOD strip ()
        RETURNS addr_2_line SELF AS RESULT
        EXTERNAL NAME 'removeLeadingBlanks';
    GRANT USAGE ON TYPE addr_2_lineTO admin;
   END INSTALL",
  "BEGIN REMOVE
     REVOKE USAGE ON TYPE addr_2_line FROM admin RESTRICT;
     DROP TYPE addr_2_line RESTRICT;
     REVOKE USAGE ON TYPE addr FROM PUBLIC RESTRICT;
     DROP TYPE addr RESTRICT;
  END REMOVE"
}
```

I.9 Using Java classes as data types

After you have installed a set of Java classes with the `SQLJ.INSTALL_JAR` procedure, and executed the appropriate SQL CREATE statements to specify SQL types defined on the Java classes, you can specify those external Java data types as the data types of SQL columns. For example:

```
CREATE TABLE emps (
    name         CHARACTER VARYING(30),
    home_addr    addr,
    mailing_addr addr_2_line
)
```

In this table, the `name` column is an ordinary SQL character string, and the `home_addr` and `mailing_addr` columns are instances of the external Java data types.

SQL columns whose data types are external Java data types are referred to as *SQL/JRT columns*.

Alternatively, if the implementation of this part of ISO/IEC 9075 supports typed tables as specified in [ISO9075-2], you can use the SQL type to create a typed table. Other tables can then reference the objects in the typed table. This representation allows the objects in the typed table to be shared (*i.e.*, referenced from multiple objects).

For example, you could store objects of type `addr` in a typed table `addresses` and reference them from one or more other tables:

```
CREATE TABLE addresses OF addr (
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

```
    REF IS id SYSTEM GENERATED ) ;
CREATE TABLE companies (
    name CHARACTER VARYING(100),
    address REF(addr) SCOPE addresses
) ;
CREATE TABLE emps2 (
    name         CHARACTER VARYING(30),
    home_addr    REF(addr) SCOPE addresses,
    mailing_addr addr_2_line
) ;
```

In a typed table such as `addresses`, each attribute of the type becomes a separate column of the same name in the typed table. In addition, the typed table has an implicit identifier column, which identifies a row (*i.e.*, an object) in the table. In the example above, the name of this column is `id` and the values for the column are automatically generated by the database system. [ISO9075-2] supports additional generation mechanisms for object identifiers, which can be defined through extended syntax in the CREATE TYPE statement (see Subclause 9.4, "<user-defined type definition>", and [ISO9075-2] for more details).

You can store references to the objects of the `addresses` table in columns of type `REF(addr)`. The definition for these columns also identifies the `addresses` table as the scope of the reference column.

I.10 SELECT, INSERT, and UPDATE

After you have specified SQL/JRT columns such as `emps.home_addr` and `emps.mailing_addr`, the values that you assign to those columns shall be Java instances. Such instances are initially generated by calls to constructor methods, using the `NEW` operator as in Java. For example:

```
INSERT INTO emps VALUES ( 'John Doe', NEW addr(), NEW addr_2_line() )
INSERT INTO emps VALUES ( 'Bob Smith', NEW addr('432 Elm Street', '95123'),
                         NEW addr_2_line('PO Box 99', 'attn: Bob Smith', '99678') )
```

The initial values specified for the SQL/JRT columns are the results of constructor method invocations. Note the use of the `NEW` keyword, whose role is the same in the facilities of this part of ISO/IEC 9075 as in Java.

Values of such columns can also be copied from one table to another. For example, assume the following additional table:

```
CREATE TABLE trainees (
    name         CHARACTER(30),
    home_addr    addr,
    mailing_addr addr_2_line
);
INSERT INTO emps
      ( SELECT * FROM trainees
        WHERE name IN ('Bill Baker', 'Chuck Morgan', 'Frank Jones') ) ;
```

Inserting objects into typed tables uses the same syntax as for regular base tables. For example:

```
INSERT INTO addresses
    VALUES ('1357 Ocean Blvd.', '99111')
```

Reference values can be obtained either directly from the referenced table (using the identifier column), or from other reference columns. For example, the following statement obtains a reference value stored in the

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

companies table and inserts it into the emps2 table. This results in a situations where the addr object is "shared" by multiple referencing parties, thereby avoiding multiple redundant copies of the same addr object.

```
INSERT INTO emps2
    VALUES ( 'Rob White , NEW addr( '165 Oak Street', '95234' ),
              ( SELECT address FROM companies
                WHERE name = 'eBiz Unlimited' ) )
```

I.11 Referencing Java fields and methods in SQL

You can invoke the methods andreference and update the fields of SQL/JRT columns such as emps.home_addr and emps.mailing_addr using SQL field qualification.

```
SELECT home_addr.to_string() , mailing_addr.to_string()
FROM emps
WHERE name = 'Bob Smith';
SELECT name, home_addr.zip_attr
FROM emps
WHERE home_addr.street_attr= '456 Shoreline Drive';
UPDATE emps
    SET home_addr.street_attr = '457 Shoreline Drive',
        home_addr.zip_attr = '99323'
WHERE home_addr.to_string() LIKE '%456%Shore%';
```

You can also access columns of objects in typed tables and invoke methods on objects in typed tables through references by using the dereference operator ("->").

```
SELECT name, mailing_addr->to_string()
FROM emps2
WHERE name = 'Bob Smith';
SELECT name, mailing_addr->street_attr
FROM emps2
WHERE mailing_addr->zip_attr = '99111';
```

I.12 Extended visibility rules

We have now defined SQL data types on the Java classes Address and Address2Line, and shown how you can use those classes as the data types of SQL columns.

Defining those SQL data types on the Java classes has one additional effect. Those SQL data types and the Java classes that they are defined upon are now added to the list of corresponding Java and SQL data types, so that we can now use Java methods whose data types are those Java classes. For example:

```
public class Utility {
    // A function version of the removeLeadingBlanks method of Address.
    public static Address stripLeadingBlanks(Address a) {
      return a.removeLeadingBlanks() ;
    }
    // A function version of the removeLeadingBlanks method of Addr2Line.
    public static Addr2Line stripLeadingBlanks(Addr2Line a) {
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      return a.removeLeadingBlanks() ;
    }
}
CREATE FUNCTION strip(a addr) RETURNS addr
    LANGUAGE JAVA PARAMETER STYLE JAVA
    EXTERNAL NAME 'address_classes_jar:Utility.stripLeadingBlanks';
CREATE FUNCTION strip(a addr_2_line) RETURNS addr_2_line
    LANGUAGE JAVA PARAMETER STYLE JAVA
    EXTERNAL NAME 'address_classes_jar:Utility.stripLeadingBlanks';
```

Note that the CREATE FUNCTION statement has no syntax to indicate that the referenced method specifies SELF AS RESULT. Because the referenced methods have that specification, the two `strip` functions both return copies of their input parameters.

I.13 Logical representation of Java instances in SQL

We saw in Subclause I.10, "SELECT, INSERT, and UPDATE", that the values assigned to such SQL/JRT columns are assigned from other SQL/JRT columns or from the results of calling Java constructors or other methods. Hence, the values assigned to SQL/JRT columns are ultimately derived from values constructed by Java methods in the JVM. Such values are represented in SQL/JRT columns by a value that is obtained from either the Java interface `java.io.Serializable` or the Java interface `java.sql.SQLData`. One or both of those interfaces shall be implemented by a Java class that is used as a data type in SQL. The value obtained from that interface is effectively a copy of the Java instance.

For example:

```
INSERT INTO emps
    VALUES ( 'Don Green', NEW addr('234 Stone Road', '99777'),
             NEW addr_2_line() );
```

The `addr` constructor method with the NEW operator constructs an `addr` instance and returns a reference to it. However, since the target is an SQL/JRT column, the SQL system uses the interface `java.io.Serializable` or `java.sql.SQLData` to obtain data that is effectively a copy of the new Java value, and copies that value into the new row of the `emps` table.

The `addr_2_line` constructor method operates the same way as the `addr` method, except that it returns a default instance rather than an instance with specified parameter values. The action taken is, however, the same as for the `addr` instance.

Note that the values stored into SQL/JRT columns are copies of Java instances, not references. For example:

```
INSERT INTO emps (name, home_addr)
    VALUES ('Sally Green',
            SELECT home_addr
            FROM emps e2
            WHERE e2.name='Don Green');
```

This INSERT statement copies the `home_addr` column from the 'Don Green' row to the new 'Sally Green' row. Note that the column value, which contains a copy of the Java instance, is itself copied. Thus, the `home_addr` columns of the 'Sally Green' row and the 'Don Green' row are independent copies, not references to a shared copy. In particular, the following statement has no effect on the 'Sally Green' `home_addr`:

```
UPDATE emps
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    SET home_addr.zip_attr = '94608'
WHERE name = 'Don Green';
```

The values stored in SQL/JRT columns are "reassembled" when a column is passed as a parameter to a function that is defined on a Java method. For example:

```
UPDATE emps
    SET home_addr = strip(home_addr)
WHERE SUBSTRING(home_addr.street_attr, 1, 1) = ' ';
```

The `strip` function is an SQL function defined on the Java static method `Utility.stripLeadingBlanks`. The parameter data type of the function is the `addr` data type. When we pass the `home_addr` column as an argument, the value in the current row is reassembled into the JVM, and a reference to the reassembled value is passed to the method `Utility.stripLeadingBlanks`. The result of that function is of data type `Address`, which corresponds with the SQL data type `addr`. The Java interface `java.io.Serializable` or `java.sql.SQLData` is applied to this returned value, and the result is copied back into the column.

Finally, consider the role of SQL nulls. For example:

```
INSERT INTO emps (name)
    VALUES ('Mike Green');
```

The INSERT statement specifies no values for the `home_addr` or `mailing_addr` columns, so those columns will be set to the null value, in the same manner as any other SQL column whose value is not specified in an INSERT. This null value is generated entirely in SQL, and initialization of the `mailing_addr` column does not involve the JVM at all.

I.14 Static methods

The methods of a Java class can be specified as either STATIC or non-STATIC. For example, in the `Address` class, the `toString` method is non-STATIC and the `contiguous` method is STATIC.

The METHOD clauses of SQL CREATE TYPE statements can also specify that a method is STATIC or non-STATIC. For example, the CREATE TYPE for the `addr` SQL type specifies that `to_string` is a non-STATIC method and `contiguous` is a STATIC method.

In Java and SQL, a non-STATIC method is referenced by qualification on an instance of the class/type. For example, assume that *JAI* and *SAI* are respectively Java and SQL variables of type/class `Address` or `addr`. You would reference the `toString` or `to_string` methods of those instances by the expressions `JAI.toString()` or `SAI.to_string()`.

In Java, a STATIC method can be referenced by qualification on *either* the class or on an instance of the class. For example, you can reference the `contiguous` method as either `Address.contiguous(...)` or as `JAI.contiguous(...)`.

In SQL, a STATIC method is referenced by qualification on the type, not on an instance. For example, you reference the `contiguous` method as `addr::contiguous(...)`. You cannot reference the SQL `contiguous` method as (for example) `SAI.contiguous(...)`. Note that in SQL, static method qualification on the type name specifies a <double colon> as the qualification punctuation, rather than a single <period>. This avoids ambiguities with other SQL constructs.

NOTE 76 — In addition to referencing static methods by such field qualification, you can also reference static methods by specifying standalone procedures or functions, using the SQL routines facilities of this part of this International Standard. For example:

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
CREATE FUNCTION contig_function (A1 addr, A2 addr)
    RETURNS CHARACTER(3)
  LANGUAGE JAVA PARAMETER STYLE JAVA
  EXTERNAL NAME 'address_classes_jar:Address.contiguous';
```

I.15 Static fields

The fields of a Java class can be specified as either STATIC or non-STATIC. In the example `Address` class, the `street` and `zip` fields are non-STATIC and the `recommendedWidth` field is STATIC.

The static fields of a Java class can be specified as FINAL, which makes them read-only. Non-FINAL fields can be updated. Users do not always specify the FINAL clause for read-only static fields.

The SQL CREATE TYPE does not include a facility for specifying atributes to be STATIC. This is because of the difficulty in specifying what the scope, persistence, and transactional properties of static fields would be in a database environment.

The SQL CREATE TYPE does, however, provide a shorthand mechanism for read-only access to the values of Java static fields. This is illustrated in the CREATE TYPE for `addr`, which specifies a STATIC METHOD clause for the `recommendedWidth` field:

```
CREATE TYPE addr EXTERNAL NAME 'address_classes_jar:Address'
    LANGUAGE JAVA
    USING SERIALIZABLE
    AS (
  zip_attr CHARACTER(10) EXTERNAL NAME 'zip',
  street_attr CHARACTER VARYING(50) EXTERNAL NAME 'street' )
  STATIC METHOD rec_width () RETURNS INTEGER
    EXTERNAL VARIABLE NAME 'recommendedWidth',
  ... ;
```

The STATIC METHOD clause for `rec_width` specifies that it is an integer-valued method with no parameters. The EXTERNAL clause for a static method would normally specify the name of a static method of the Java class. In this case, however, the EXTERNAL clause specifies the keyword VARIABLE, and gives the name of a static field of the Java class. When a STATIC METHOD clause specifies EXTERNAL VARIABLE, the method shall have no parameters, and the specified Java name shall be that of a static field. Such a static method is invoked in the normal manner, and returns the value of the specified Java static field.

Given such a declaration, you can reference the `rec_width` method in the same manner as other static methods, and access the `recommendedWidth` field:

```
SELECT *
FROM emps
WHERE LENGTH(home_addr.street_attr) > addr::rec_width();
```

SQL provides no way to update the values of Java static fields.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

I.16 Instance-update methods

A non-static Java class method is invoked by qualification on an instance of the class. For example, assuming that JAI is an instance of the Java `Address` class, you would reference the `toString` or `removeLeadingBlanks` methods as `JAI.toString()` or `JAI.removeLeadingBlanks()`.

Such non-static methods generally reference the fields of the instance that qualifies the method reference, *e.g.*, the instance JAI. The `toString` method references the instance JAI in a read-only manner, returning a string representation of that instance. The `removeLeadingBlanks` method, however, references the qualifying instance in a manner that updates the value of the instance. That update is intended to be a side-effect of the method invocation.

Read-only methods such as `toString` fit naturally into SQL. For example, given the above `emps` table:

```
SELECT name, home_addr.to_string()
FROM emps
WHERE home_addr.to_string() <> x;
```

As described in Subclause I.13, "Logical representation of Java instances in SQL", Java instances stored in SQL columns and variables are copies of the Java values, not references to such values. Therefore, methods such as `removeLeadingBlanks` that have side-effects on the qualifying instances do not fit naturally into the SQL framework. For this reason, the SQL CREATE TYPE for a Java class provides a special mechanism for referencing Java methods that have side effects. This is illustrated by the METHOD clause for `remove_leading_blanks`:

```
CREATE TYPE addr EXTERNAL NAME 'address_classes_jar:Address'
  LANGUAGE JAVA
  USING SERIALIZABLE
  AS ...
  METHOD remove_leading_blanks () RETURNS addr SELF AS RESULT
    EXTERNAL NAME 'removeLeadingBlanks';
```

Recall that the `removeLeadingBlanks` method of the Java Address class is a `void` method. You might therefore expect to specify the SQL `remove_leading_blanks` as a `void` method, that is, a "procedure method". However, the SQL CREATE TYPE does not provide a way to specify `void` methods or procedure methods. This is because such methods would almost always perform side effects on the qualifying instance, and would therefore not be suitable for a value-oriented SQL context.

The SQL `remove_leading_blanks` method specifies the clause RETURNS SELF AS RESULT. This clause has the following significance:

— The return type of the method is defined to be the containing SQL data type. That is, the SQL `remove_leading_blanks` method is an `addr`-valued method. This is the case irrespective of the return type of the underlying Java method. In the typical case, the underlying Java method will be a `void` method, but as we will discuss below, this is not required.

— At runtime, the specified Java method is invoked in the normal manner, and updates the fields of a copy of the qualifying instance. When the invocation is complete, the SQL system then makes a copy of the updated value of the qualifying instance, and returns that copy as the result of the method.

As example invocation of `remove_leading_blanks` is as follows:

```
UPDATE emps
    SET home_addr = home_addr.remove_leading_blanks()
WHERE ... ;
```

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Such an UPDATE statement proceeds in the normal manner to process each row of the emps table, and to perform the SET actions in each row for which the WHERE clause is true. For such a row, the value of the home_addr column is passed to the Java virtual machine, which evaluates the removeLeadingBlanks method for that instance of the Address class. That method performs side effects on the fields of that copy of the current home_addr column, and returns. The SQL system then makes a copy of that updated value of the Address instance, and returns that copy as the result of the call to remove_leading_blanks. That copy is then assigned back to the home_addr column of the current row.

Consider a somewhat different invocation of remove_leading_blanks:

```
SELECT name, home_addr.remove_leading_blanks().street_attr
FROM emps
WHERE ... ;
```

This SELECT statement processes the emps rows, and evaluates the select-list for selected rows. The second element of that select-list invokes the remove_leading_blanks method of the home_addr column. As above, this invocation passes a copy of the home_addr value to the JVM, where the removeLeadingBlanks method updates the copy. The SQL system then returns a copy of that updated copy, and extracts the street_attr attribute. That street_attr attribute will reflect the removal of leading blanks that has been done. However, these actions do not affect the value of the home_addr column in the emps table.

This SELF AS RESULT mechanism provides a general way for SQL to apply the side-effects of arbitrary Java methods.

Java methods that update the qualifying instance will commonly be written as void methods. In some cases, however, such methods are written to return (for example) integer values that provide some sort of status feedback, such as an “OK” indication. For this reason, you can specify the RETURNS SELF AS RESULT clause for arbitrary Java methods, irrespective of the return type of the method. Note, however, that this return value that the method invocation explicitly provides is simply discarded by the SQL system, which replaces that explicit returned value with the implicit copy of the qualifying instance.

I.17 Subtypes in SQL/JRT data

Recall the example Java classes Address and Address2Line , and the corresponding SQL data types addr and addr_2_line. The Address2Line class is a subclass of the Address class, so you can make use of the substitutability and method overloading characteristics of Java.

For example, you can assign addr_2_line values to addr columns. We can illustrate this with the emps table, in which the home_addr column is an addr and the mailing_addr column is an addr_2_line:

```
UPDATE emps
    SET home_addr = mailing_addr
WHERE home_addr IS NULL;
```

For the rows in which we perform the above SET clause, the home_addr column will contain an addr_2_line, even though the declared type of home_addr is addr.

Such an assignment does not modify the actual instance value or its runtime data type. Thus, when you store addr_2_line values from the mailing_addr column into the home_address column, those values still have the run-time type of addr_2_line. The effect of this can be seen in the following example.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Recall that the addr type and the addr_2_line subtype both have a method named toString, which returns a String form of the complete address data.

Consider the following call ofthe to_string method:

```
SELECT name, home_addr.to_string()
FROM emps
WHERE home_addr.to_string() NOT LIKE '%Line2=%';
```

For each row of emps, the declared type of the home_addr column is addr, but the runtime typc of the *home_addr* value will be either addr or addr_2_line, depending on the effect of the previous UPDATE statement. For rows in which the runtime value of the home_addr column is an addr, the to_string method of the addr class will be invoked, and for rows in which the runtime value of the home_addr column is an addr_2_line, the to_string method of the addr_2_line subclass will be invoked.

The way that this runtime selection of the to_string method is performed is as follows:

— At compile time, the SQL system determines that the calls of home_addr.to_string() are syntactically correct, and that the result type is suitable (*e.g.*, for the LIKE predicate).

— At runtime, the SQL system will process the calls of home_addr.to_string() for each row of emps in the following steps:

  • The value of the home_addr column for the row is reassembled into the JVM, and a reference *R* for that reassembled value is obtained.

  • The invocation *R*.toString() is passed to the JVM for evaluation. The JVM performs the run time selection of the appropriate toString method, and returns the result.

I.18 References to fields and methods of null instances

Assume that you insert the following row into the emps table:

```
INSERT INTO emps (name)
VALUES ('Charles Green')
```

Note that the home_address and mailing_address columns are both null, since no values were specified for them.

Consider the following SELECT statement:

```
SELECT name, home_addr.zip_attr
FROM emps
WHERE home_addr.zip_attr IN ('95123', '95125', '95128');
```

The intention of this SELECT is to retrieve the given values of those emps rows for which the zip field of home_addr has one of the specified values. This would not include the rows of emps for which home_addr is null.

When we execute this SELECT statement, the WHERE clause will be evaluated for each row of emps, including the rows in which the home_addr column is null. In Java, and other programming languages, if you attempt to reference a field of a null instance, an exception condition is raised. If we use that rule in SQL, then the above SELECT would raise an exception if the home_addr column in any row of emps were null.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Note that this is an exception for the entire SELECT statement, not for particular rows. To get the desired effect, we would have to write the SELECT as follows:

```
SELECT name, home_addr.zip_attr
FROM emps
WHERE CASE
        WHEN home_addr IS NOT NULL
          THEN home_addr.zip_attr
        ELSE NULL
      END IN ('95123', '95125', '95128');
```

In fact, if we specify that field references to null instances raise an exception, then virtually all WHERE clause references to fields would have to be written with such a CASE expression. This would be exceedingly tedious, so the SQL/JRT rule for field references to null instances is different from Java:

If the value of the instance specified in a field reference is null, then the field reference is null.

This rule is equivalent to specifying that the above CASE expression is implicit.

This rule therefore allows you to write the SELECT in the original form. For rows whose `home_addr` column is null, the field reference `home_addr.zip_attr` will be null.

This rule for field references with null instances only applies to field references in “value”, or “right-hand-side” contexts, not to field references that are targets of assignments or SET clauses.

For example:

```
UPDATE emps
    SET home_addr.zip_attr = '99123'
WHERE name = 'Charles Green';
```

This WHERE clause will obviously be true for the 'Charles Green' row, so the UPDATE statement will try to perform the given SET clause. This will raise an exception, since you cannot assign a value to a field of a null instance. This is because the null instance has no field to which a value can be assigned.

In other words, field references to fields of null instances are valid and return the null value in right-hand-side contexts, and cause exceptions in left-hand-side contexts.

Exactly the same considerations apply to invocations of methods of null instances, and the same rule is applied.

For example, suppose that we modify the previous example and invoke the `to_string` method of the `home_addr` column:

```
SELECT name, home_addr.to_string()
FROM emps
WHERE home_addr.to_string() = 'Street=234 Stone Road ZIP=99777'
```

If we apply the strict Java rule, then invocations of the `to_string` method for rows in which the `home_addr` column is null will raise an exception. We would therefore, as above, need to code the SELECT as follows:

```
SELECT name, home_addr.to_string()
FROM emps
WHERE CASE
        WHEN home_addr IS NOT NULL
         THEN home_addr.to_string()
        ELSE NULL
      END = 'Street=234 Stone Road ZIP=99777';
```

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

We therefore extend the Java rule for method invocation in the same manner that we extended the Java rule for field references:

If the value of the instance specified in an instance method invocation is null, then the result of the invocation is null.

I.19 Ordering of SQL/JRT data

In an earlier clause, we created the `emps` table, with columns `home_addr` and `mailing_addr` whose data types are declared to be the Java classes, respectively, `Address` and `Address2Line`. Now suppose that you reference those columns in statements such as the following:

```
SELECT DISTINCT *
FROM emps E1, emps E2
WHERE E1.home_addr = E2.home_addr
  AND E1.mailing_addr > E2.mailing_addr
UNION
SELECT DISTINCT *
FROM emps E1, emps E2
WHERE E1.mailing_addr = E2.mailing_addr
  AND E1.home_addr > E2.home_addr
GROUP BY home_addr
ORDER BY home_addr, mailing_addr;
```

This statement involves numerous references to `home_addr` and `mailing_addr` that imply ordering relationships:

1) The DISTINCT keyword is defined in terms of equality of rows, which is specified as a pairwise comparison of corresponding columns. That is, to determine if two rows of emps are DISTINCT, you have to compare their respective `home_addr` and `mailing_addr` columns.

2) The direct comparisons using "=" and ">", *etc.* all require ordering properties.

3) The UNION operator doesn't specify UNION ALL, so it will eliminate duplicates. This will require the same kind of comparisons as the DISTINCT clause.

4) The GROUP BY requires partitioning the rows into sets with equal values of the grouping column.

5) The ORDER BY requires determination of the ordering properties of the order columns.

When you create an external Java data type with a CREATE TYPE...EXTERNAL LANGUAGE JAVA statement, the new external Java data type has no ordering capability. That is, its "ordering form" is "none". Instances of an external Java data type whose ordering form is none cannot be used in any of the above ordering relationships.

To define ordering for an external Java data type, you use the CREATE ORDERING statement:

```
<user-defined ordering definition> ::=
    CREATE ORDERING FOR <user-defined type name>
    <ordering form>

<ordering form> ::=
    EQUALS ONLY BY <ordering category>
  | ORDER FULL BY <ordering category>
```

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<ordering category> ::=
    MAP WITH <ordering routine>
  | RELATIVE WITH <ordering routine>
  | RELATIVE WITH COMPARABLE INTERFACE
  | STATE
```

The significance of the EQUALS ONLY and FULL alternatives is as follows:

— EQUALS ONLY specifies that instances of the associated class can be referenced in equals (=) and not equals (<>) operations, SELECT DISTINCT, UNION with duplicate elimination, and GROUP BY, but not in other ordering contexts.

— FULL specifies that instances of the associated class can be referenced in any ordering context.

The STATE clause specifies that instances will be ordered on the values of the attributes of the type.

The MAP clause specifies the name of a method or function that will map instances of the associated class to values of some built-in SQL data type, whose ordering defines the ordering of the associated class. The map routine needn't define a 1:1 into correspondence. It can map distinct instance values to the same result. This would be done in order to equate 6/8 and ¾ for a class that implements rational numbers. It can also be done for folded comparisons, and other cases where it is desirable to equate distinct instances.

The RELATIVE WITH <ordering routine> clause specifies the name of a method or function that compares instances of the associated class and returns an integer result. The runtime result value for two instances X and Y is -1, 0, or +1 to indicate respectively that X is *less than*, *equal to*, or *greater than* Y.

The RELATIVE WITH COMPARABLE INTERFACE clause may be used only in orderings for SQL data types whose subject Java class implements `java.lang.Comparable`. The `int compareTo` method of the subject Java class determines the relative ordering for two instances X and Y, returning a negative integer, 0 (zero), or a positive integer to indicate respectively that X is *less than*, *equal to*, or *greater than* Y.

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Bibliography

[1] [RFC2368] RFC 2368, *The mailto URL scheme*, P. Hoffman, L. Masinter, J. Zawinski, July, 1998
http://www.ietf.org/rfc/rfc2368.txt

[2] [RFC3986] RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter, January, 2005
http://www.ietf.org/rfc/rfc3986.txt

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(Blank page)

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Index

Index entries appearing in **boldface** indicate the page where the word, phrase, or BNF nonterminal was defined; index entries appearing in *italics* indicate a page where the BNF nonterminal was used in a Format; and index entries appearing in roman type indicate a page where the word, phrase, or BNF nonterminal was used in a heading, Function, Syntax Rule, Access Rule, General Rule, Leveling Rule, Table, or other descriptive text.

— A —

<action group> • **94**
ADA • 108
ALL • 187
AND • 97, 98, 100, 101, 108, 114, 146, 147, 149, 187
ARRAY • 54
array mappable • *14*, 55
AS • 17, 58, *63*, 64, 68, 97, 98, 100, 101, 103, 104, 145, 146, 147, 149, 172, 173, 174, 176, 177, 180, 182, 183, 184
associated JAR • *86*, *88*
attempt to remove uninstalled JAR • 89, 117
attempt to replace uninstalled JAR • 87, 117
<attribute definition> • **67**, 68, 69, 128

— B —

BEGIN • *94*, 95, 163, 176, 177
block • **5**, *9*, 127
BOTH • 85, 87, 89, 91
BY • 146, 147, 149, 187, 188

— C —

CALL • 59, 145, 158
call type data item • *35*, 44, *46*, 47, 49, 50
CALLED • 157
CASE • 156, 157, 186
CAST • 68
CATALOG_NAME • 97, 98, 105, 110, 112
CHARACTER • 54, 85, 87, 89, 91, 128, 129, 142, 144, 145, 148, 154, 155, 157, 158, 163, 172, 173, 174, 176, 177, 178, 182
CHARACTER_SET_CATALOG • 103, 104
CHARACTER_SET_NAME • 103, 104
CHARACTER_SET_SCHEMA • 103, 104
CHECK • 105, 108, 110, 112, 114
class • 6, *9*, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 26, 29, 32, 33, 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50, 52, 53, 54, 55, 56, 57, 64, 65, 68, 69, 74, 79, 86, 87, 88, 90, 93, 117, 128, 131, 141, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 164, 165, 166, 169, 170, 171, 172, 173, 174, 175, 179, 180, 181, 182, 183, 184, 185, 188
class declaration • **5**, *9*
class file • **5**, *10*
<class identifier> • **25**, 26, *32*, 33, 65, 127
class instance • **5**, *9*
class method • *9*
class name resolution • *12*
class variable • **5**, *9*
close call • *46*, *49*, *50*
COBOL • 108
COLLATION_CATALOG • 103, 104
COLLATION_NAME • 103, 104
COLLATION_SCHEMA • 103, 104
<command> • **94**, 95
COMMIT • 158
COMPARABLE • 18, *23*, 29, *79*, 114, 188
<comparable category> • **79**
comparison function • *18*, 29
CONSTRAINT • 105, 107, 108, 110, 112, 114
CONSTRUCTOR • 17, 18, 55, 57, 172, 173, 174, 176, 177
CONTAINS • 75, 145
corresponding Java data type • *13*
corresponding Java field name • *67*
corresponding Java method name • *58*
CREATE • 16, 97, 98, 100, 101, 103, 105, 107, 110, 112, 141, 142, 144, 145, 147, 148, 150, 151, 152, 153, 154, 155, 157, 158, 159, 160, 162, 163, 164, 165, 166, 172, 173, 174, 175, 176, 177, 178, 180, 181, 182, 183, 187
CURRENT_PATH • 12
CURRENT_USER • 97, 98, 100, 101

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— D —

DATA • 75, 145, 148, 150, 151, 155, 163
data exception • 69
Data type conversion tables • *13*
DATE • 152
DECIMAL • 142, 148, 163
default connection • **6**, *93*, 130
DELETE • 145
dependent JARs • *90*
dependent privilege descriptors still exist • 83
dependent SQL routines • *90*
dependent SQL routines • *88*
dependent SQL types • *88*, *90*
deployed routines • *95*
deployed types • *95*
deployment descriptor • iii, 6, 22, 86, 89, 127, 144, 159, 163, 164, 172, *176*
deployment descriptor file • *6*, 7, 94, 95, 163, 164, 176
DESC • 146, 147, 149
<descriptor file> • 66, 72, 75, 78, 79, 80, 81, 84, **94**, 121, 122
direct subclass • *9*, 64
direct superclass • *9*, 175
DISTINCT • 187, 188
DOUBLE • 152
DROP • 141, 158, 159, 162, 163, 164, 176, 177
DYNAMIC • 37, 51, 150, 151, 155, 163

— E —

effective SQL parameter list • 11, 12, 35, 51, 52, 54, *74*, 75
ELSE • 156, 186
END • *94*, 95, 156, 163, 164, 177, 186
EQUALS • 187, 188
executable form • *11*
EXECUTE • 21, 95, 127, 163, 164
extend • *9*, 12, 187
EXTERNAL • 20, *63*, *67*, 145, 148, 150, 151, 152, 153, 154, 155, 156, 157, 158, 163, 172, 173, 174, 176, 177, 180, 182, 183, 187
<external Java attribute clause> • 19, 20, **67**
<external Java class clause> • 19, **63**, 64, 112, 114
external Java data type • 6, 14, *15*, 16, 17, 27, 28, 29, 52, 61, 62, *64*, 65, 67, 68, 69, 71, 79, 83, 115, 123, 127, 128, 187
external Java data types • 6, 15, 16, 20, 21, 64, *173*, 177
<external Java method clause> • **63**, 64, 65, 108
<external Java reference string> • 51, 53, **73**, 75, 83, 88, 90, 110, 160
external Java routine • **6**, 11, 12, 14, 34, 35, 39, 40, 42, 44, 46, 47, 48, 49, 51, *73*, 74, 75, 77, 78, 83, 108, 110, 121, 124, 127, 128, 131, 167
<external Java type clause> • 6, 19, 52, **63**, 64, 66, 95, 121, 124
external routine • 6, 11, 34, 39, *73*
external routine invocation exception • 37, 44
<external variable name clause> • 18, **63**

— F —

Feature F391, “Long identifiers” • 103
FETCH • 156
fetch call • *46*, *47*
field • **5**
fields • ix, ix, vi, *9*, 18, 19, 57, 65, 141, 172, 179, 182, 183, 184, 186, 187
FINAL • 182
final • *18*
FOR • 187
FOREIGN • 105, 107, 110, 112
FORTRAN • 108
FROM • 85, 87, 89, 91, 97, 98, 100, 101, 103, 104, 105, 110, 112, 113, 145, 146, 147, 149, 155, 156, 157, 158, 161, 164, 177, 178, 179, 180, 182, 183, 184, 185, 186, 187
FULL • 187, 188
FUNCTION • 6, 78, 141, 144, 145, 152, 153, 154, 155, 157, 158, 159, 160, 162, 163, 164, 165, 166, 173, 180, 182

— G —

GENERAL • 12, 75
GENERATED • 178
GRANT • 97, 98, 100, 101, 141, 144, 159, 162, 163, 164, 176, 177
GROUP • 187, 188

— I —

<implementor block> • *22*, **94**, 95, 130
<implementor name> • **94**, 95, 130
IN • 51, 52, 54, 55, 75, 85, 87, 89, 91, 97, 98, 100, 101, 105, 108, 110, 112, 114, 178, 185, 186
INOUT • 51, 52, 54, 55, 56, 75, 76, 123, 141, 146, 148
INPUT • 157, 158
INSERT • 145, 178, 179, 180, 181, 185
INSTALL • *94*, 163, 176, 177
<install actions> • 6, *22*, 85, 86, **94**, 95, 129, 130
installed JAR • **6**, *16*
INSTANCE • 55, 57
instance initializer • **5**

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

instance method • *9*
instance variable • **5**, *9*
INTEGER • 85, 89, 142, 145, 148, 150, 151, 152, 153, 154, 155, 157, 163, 172, 173, 174, 176, 182
INTERFACE • *23*, *79*, 188
interface • **5**, *10*, 13, 14, 16, 19, 20, 29, 36, 37, 40, 41, 42, 45, 46, 47, 48, 49, 64, 68, 69, 127, 169, 180, 181
<interface specification> • 16, 19, 20, 35, 37, 40, 41, 45, 46, 48, 49, 61, 62, **63**, 64, 66, 67, 68, 69, 115, 123, 127, 128
<interface using clause> • 16, 20, **63**, 64, 128
INTO • 178, 179, 180, 181, 185
invalid class deletion • 88, 90, 117
invalid JAR name • 85, 87, 89, 91, 117
invalid JAR name in path • 32, 117
invalid JAR removal • 90, 117
invalid path • 91, 117
invalid replacement • 88, 117
invalid URL • 85, 87, 117, 129
IS • 108, 114, 146, 147, 149, 156, 178, 184, 186

— J —

Feature J511, “Commands” • 22, 66, 72, 75, 78, 79, 80, 81, 84, 119, 121, 122
Feature J521, “JDBC data types” • 76, 122
Feature J531, “Deployment” • 22, 59, 86, 90, 96, 119, 122
Feature J541, “SERIALIZABLE” • 66, 119, 122, 123
Feature J551, “SQLDATA” • 66, 119, 123
Feature J561, “JAR privileges” • 82, 119, 123
Feature J571, “NEW operator” • 28, 123, 127
Feature J581, “Output parameters” • 76, 123
Feature J591, “Overloading” • 66, 123
Feature J601, “SQL-Java paths” • 33, 92, 123
Feature J611, “References” • 42, 123, 124
Feature J621, “external Java routines” • 76, 78, 119, 124
Feature J622, “external Java types” • 66, 72, 79, 80, 119, 124
Feature J631, “Java signatures” • 31, 124
Feature J641, “Static fields” • 66, 124, 125
Feature J651, “SQL/JRT Information Schema” • 98, 99, 102, 104, 125
Feature J652, “SQL/JRT Usage tables” • 97, 100, 101, 125, 126
JAR • 6, 7, 10, 11, 12, 21, 22, *23*, 26, 32, 54, 56, 57, 59, 64, 74, 75, *82*, 83, 85, 86, 87, 88, 89, 90, 91, 92, 94, 95, 97, 98, 105, 106, 107, 110, 112, 113, 117, 129, 143, 144, 145, 147, 150, 152, 153, 159, 160, 162, 163, 164, 165, 166, 167, 171, 173, 176
<jar and class name> • 13, 14, 16, **25**, 52, 53, 64, 65, *73*, 83, 114, 160
<jar id> • 12, **25**, 26, 64, 85, 87, 89, 91, 105, 106, 107, 110, 112
<jar name> • **25**, 26, *32*, 51, 52, 53, 59, 74, 81, *82*, 83, 84, 85, 87, 88, 89, 90, 91, 95, 105, 106, 107, 110, 112, 122, 123
JAVA • 6, 12, 15, *23*, 34, 41, *43*, *63*, 64, 65, *73*, 74, 75, 76, 95, 108, 111, 114, 115, 121, 123, 124, 145, 148, 150, 151, 152, 153, 154, 155, 156, 157, 158, 163, 172, 173, 174, 176, 177, 180, 182, 183, 187
Java archive • *10*, 21
Java Archive (JAR) • **6**
<Java class name> • 13, 14, 17, **25**, 26, 33, 51, 52, 53, 64, 74
<Java data type> • **31**, 55, 56, 57, 76, 122
Java DDL • 85, 87, 88, 89, 90, 91, 117, 118, 129
Java execution • 32, 33, 117, 118
<Java field name> • 17, **25**, 26, 52, 53, 67, 68, 69, 127
Java file • *10*
<Java identifier> • **25**, 52, 53, 65
<Java method and parameter declarations> • 52, **63**, 65, 108
<Java method name> • **25**, 26, 51, 52, 53, 58, *63*, 65, *73*, 74, 127
Java method signature • *9*
<Java parameter declaration list> • **31**, 34, 51, 52, 53, 55, 57, *63*, 64, 65, *73*, 75, 124, 127, 128
<Java parameters> • **31**, 55, 56
Java Virtual Machine • **5**
JDBC ResultSet • *13*
JOIN • 97, 100, 101
JVM • **6**

— K —

KEY • 105, 107, 110, 112

— L —

LANGUAGE • 6, *63*, 65, 73, 75, 145, 148, 150, 151, 152, 153, 154, 155, 156, 157, 158, 163, 172, 173, 174, 176, 177, 180, 182, 183, 187
<language name> • 17, **43**, 65, 75, 76, 121, 123, 124
LENGTH • 182
LIKE • 179, 185
LN • 45
local variable • **5**, *9*
LOCATOR • 64

— M —

manifest • 12, *21*, 86, 89, 94, 164
MAP • 18, 29, 188
mappable • *14*, 31, 42, 55, 56, 58, 160

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

matching new classes • *87*
matching old classes • *87*
METHOD • 16, *63*, 172, 173, 174, 176, 177, 181, 182, 183
method • *9*, 10, 11, 14, 15, 16, 17, 18, 19, 21, 27, 28, 29, 31, 34, 35, 36, 37, 38, 39, 40, 41, 42, 45, 46, 47, 48, 49, 50, 51, 53, 54, 55, 56, 57, 58, 64, 68, 69, 74, 80, 88, 93, 108, 109, 114, 124, 127, 128, 131, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 165, 166, 169, 170, 171, 173, 174, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188
<method characteristic> • **63**, 64, 65
method spec descriptor • *17*
<method specification> • 17, 18, 52, **63**, 64, 66, 123
method specification descriptor • *17*
MODIFIES • 75, 145, 163
MUMPS • 108

— N —

NAME • *63*, *67*, 106, 142, 145, 148, 150, 151, 152, 153, 154, 155, 156, 157, 158, 163, 172, 173, 174, 176, 177, 180, 182, 183
nested class • **5**
NEW • 178, 179, 180
new classes • *87*, 88, 160
NO • 108, 109, 145, 163
no data • 46, 48, 49
no subclass • 117
<non-reserved word> • **23**
non-static • *18*, 183
NOT • 105, 108, 110, 112, 114, 146, 147, 149, 156, 185, 186
NULL • 68, 108, 114, 146, 147, 149, 156, 157, 158, 184, 186
null characters • *45*
null instance used in mutator function • 69
null value not allowed • 37, 44

— O —

object mappable • *13*, 14, 55, 58
<object name> • 81, **82**, 83, 84, 122, 123
OF • 177
OLB-specific error • 118
old classes • *87*, 88, 160
old JAR • *87*
ON • 97, 98, 100, 101, 157, 158, 163, 164, 176, 177
ONLY • 187, 188
OPTION • 97, 98, 100, 101
OR • 97, 98, 100, 101, 105, 108, 110, 112, 114, 145
ORDER • 146, 147, 149, 187
ORDERING • 187
<ordering category> • **79**, 187, 188
OUT • 51, 52, 54, 55, 56, 74, 76, 123, 141, 146, 148, 160, 163
output mappable • *13*, 14, 55

— P —

package • **5**, 6, *9*, 10, 12, 17, 31, 53, 93, 129, 145, 149, 151, 165, 166
<package identifier> • **25**, 26, 127
<packages> • **25**, 26, *32*, 33
PARAMETER • 34, 41, 65, 74, 75, 145, 148, 150, 151, 152, 153, 154, 155, 156, 157, 158, 163, 180, 182
<parameter style> • 17, 65, **73**
PASCAL • 108
<path element> • **32**, 33, 83, 90, 91
PLI • 108
PRECISION • 152
PRIMARY • 105, 107, 110, 112
private • *10*, 19, 57, 169
PROCEDURE • 6, 78, 141, 144, 145, 147, 148, 150, 151, 152, 153, 154, 155, 157, 158, 159, 160, 162, 163, 164, 165, 166, 173
protected • *10*
PUBLIC • 97, 98, 100, 101, 163, 164, 176, 177
public • *10*, 14, 15, 16, 19, 20, 56, 57, 142, 143, 146, 147, 149, 152, 154, 155, 158, 161, 169, 170, 171, 179

— Q —

<qualified Java field name> • **25**, 52, 65, 108

— R —

READS • 75, 145, 148, 150, 151, 155, 163
REAL • 152
REF • 178
<referenced class> • **32**
REFERENCES • 105, 107, 110, 112
RELATIVE • 18, 29, *79*, 188
relative URLs • *12*
REMOVE • *94*, 163, 164, 177
<remove actions> • 6, *22*, 89, **94**, 95, 129, 130, 163
<reserved word> • **23**
<resolution jar> • **32**, 83, 91
RESTRICT • 83, 159, 164, 177
RESULT • 17, 37, 51, 58, 73, 150, 151, 155, 163, 172, 173, 174, 176, 177, 180, 183, 184
result data items • *35*
result set cursors • *13*
result set mappable • *14*

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

result set oriented • *14*
result sets returned • 42
RETURN • 13
RETURNS • 52, *63*, 145, 152, 153, 154, 155, 157, 158, 163, 172, 173, 174, 176, 177, 180, 182, 183, 184
REVOKE • 141, 144, 162, 163, 164, 176, 177
ROLLBACK • 158
ROW • 48

— S —

Feature S201, “SQL routines on arrays” • 54
SCHEMA • 106
SCHEMA_NAME • 97
SCOPE • 178
SCOPE_CATALOG • 103
SCOPE_NAME • 103
SCOPE_SCHEMA • 103
SELECT • 97, 98, 100, 101, 103, 104, 105, 110, 112, 113, 145, 146, 147, 149, 155, 156, 157, 158, 161, 178, 179, 180, 182, 183, 184, 185, 186, 187, 188
SELF • 17, 58, 64, 172, 173, 174, 176, 177, 180, 183, 184
self-referencing path • 91, 117
ser file • **6**
SERIALIZABLE • 16, 19, 20, 36, 37, 40, 41, 45, 46, 48, 49, 61, 62, *63*, 64, 66, 67, 68, 69, 114, 123, 182, 183
Serializable • *10*
SET • 37, 143, 179, 181, 183, 184, 186
SETS • 51, 150, 151, 155, 163
simply mappable • *13*, 14, 55, 56, 58
SMALLINT • 153
SPECIFIC • *63*
SPECIFIC_NAME • 103, 110
SQL • 74, 75
<SQL Java path> • **32**, 33, 91, 105, 106, 123
SQL routine • 16, 21, *73*, 88, 90, 93, 127, 131
<SQL statement> • **94**, 95, 96, 122
<SQL token> • **94**, 95
SQL-Java path • *12*, 21, 32, 83, 86, 90, 91, 92, 107, 166, 167
SQL-Java path too long for information schema • 92, 118
SQL/JRT columns • *177*, 178, 180, 181
SQLDATA • 16, 19, 20, *23*, 36, 37, 40, 41, 42, 45, 47, 48, 49, *63*, 64, 66, 68, 69, 114, 123
SQLData • *10*
SQLJ Iterator • *13*
SQLSTATE • x, 14, 161
STATE • 18, 29, 142, 161, 188
STATIC • 17, 18, 38, 57, *63*, 172, 173, 174, 176, 181, 182
static • xi, iii, 1, *9*, 10, 14, 16, 17, *18*, 22, 32, 34, 35, 44, 53, 56, 57, 58, 65, 108, 109, 131, 142, 143, 144, 146, 147, 149, 152, 153, 154, 155, 157, 158, 161, 162, 167, 169, 170, 171, 173, 179, 181, 182
static field method • *65*
<static field method spec> • 16, 18, 52, 56, 57, 58, **63**, 64, 65, 66, 108, 125
static initializer • **5**
<static method returns clause> • **63**
STYLE • 34, 41, 65, 74, 75, 145, 148, 150, 151, 152, 153, 154, 155, 156, 157, 158, 163, 180, 182
subclass • *9*, 35, 39, 40, 46, 48, 49, 93, 117, 161, 169, 170, 175, 184, 185
subject Java class • **6**, *15*, 38, 39, *64*, *173*
subject Java class name • **6**
subject Java method • *34*, *58*
subject routine • *11*, 51
subject SQL data type • 29, *173*
subpackage • **5**
subpackages • *9*
SUBSTRING • 181
successful completion • 46, 47
superclass • *9*, 18, 65, 74, 174
SYSTEM • 178
system class • **6**

— T —

TABLE • 97, 98, 100, 101, 105, 107, 110, 112, 142, 159, 177, 178
THEN • 156, 186
TIME • 152
TIMESTAMP • 152
TO • 97, 98, 100, 101, 163, 176, 177
TRIM • 85, 87, 89, 91
TRUE • 6, 94, 164
TYPE • 141, 144, 160, 166, 172, 173, 174, 175, 176, 177, 178, 181, 182, 183, 187

— U —

UNDER • 172, 175, 177
UNION • 113, 187, 188
unknown name • 161
unmatched new classes • *87*
unmatched old classes • *87*
unresolved class name • 33, 117
UPDATE • 142, 143, 145, 179, 180, 181, 183, 184, 185, 186
USAGE • 21, 75, 82, 83, 86, 90, 91, 95, 141, 166, 167, 176, 177
<user-defined type body> • 52, **63**

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

USER_DEFINED_TYPE_CATALOG • 103, 104, 112
USER_DEFINED_TYPE_NAME • 103, 104, 112
USER_DEFINED_TYPE_SCHEMA • 103, 104, 112
USING • *63*, 97, 100, 101, 182, 183

— V —

VALUES • 178, 179, 180, 181, 185
VARCHAR • 142
VARIABLE • *63*, 172, 173, 174, 176, 182
VARYING • 54, 85, 87, 89, 91, 128, 129, 148, 154, 155, 157, 158, 163, 172, 173, 174, 176, 177, 178, 182
VIEW • 97, 98, 100, 101, 103, 159
visible • *14*, 19, 54, 56, 57, 86, 88, 90, 146, 158, 161

— W —

warning • 42, 46, 47, 92, 117
WHEN • 156, 186
WHERE • 97, 98, 100, 101, 143, 145, 146, 147, 149, 155, 156, 157, 158, 161, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187
WITH • 13, *79*, 97, 98, 100, 101, 188

 © ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ICS 35.060

Price based on 186 pages

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale