TECHNICAL REPORT

ISO/IEC TR 19075-4

First edition
2015-07-01

Information technology — Database languages — SQL Technical Reports —

Part 4: SQL with Routines and types using the Java^{TM} programming language

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

Partie 4: SQL avec des Routines et Types Utilisant le Langage de Programmation de Java^{TM}

Reference number
ISO/IEC TR 19075-4:2015(E)

Contents

Page

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

The main task of the joint technical committee is to prepare International Standards. 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.

In exceptional circumstances, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard (“state of the art”, for example), it may decide to publish a Technical Report. A Technical Report is entirely informative in nature and shall be subject to review every five years in the same manner as an International Standard.

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

ISO/IEC TR 19075-4 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

ISO/IEC TR 19075 consists of the following parts, under the general title *Information technology — Database languages — SQL Technical Reports*:

— Part 1: XQuery Regular Expression Support in SQL

— Part 2: SQL Support for Time-Related Information

— Part 3: SQL Embedded in Programs Using the Java™ Programming Language

— Part 4: SQL with Routines and Types Using the Java™ Programming Language

— Part 5: Row Pattern Recognition in SQL

NOTE 1 — The individual parts of multi-part technical reports are not necessarily published together. New editions of one or more parts may be published without publication of new editions of other parts.

Introduction

The organization of this part of ISO/IEC 19075 is as follows:

1) Clause 1, “Scope”, specifies the scope of this part of ISO/IEC 19075.
2) Clause 2, “Normative references”, identifies additional standards that, through reference in this part of ISO/IEC 19075, constitute provisions of this part of ISO/IEC 19075.
3) Clause 3, “Routines tutorial”, provides a tutorial on the use of routines written in the Java programming language within SQL expressions and statements.
4) Clause 4, “Types tutorial”, provides a tutorial on the use of user-defined types written in the Java programming language within SQL expressions and statements.

TECHNICAL REPORT ISO/IEC TR 19075-4:2015

Information technology — Database languages — SQL Technical Reports —

Part 4:
SQL with Routines and Types Using the Java™ Programming Language

1 Scope

This Technical Report provides a tutorial of SQL Routines and Types Using the Java™ Programming Language.

The Report discusses the following features of the SQL Language:

— The use of routines written in the Java programming language within SQL expressions and statements.

— the use of user-defined types written in the Java programming language within SQL expressions and statements.

(Blank page)

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 9075-1:2011, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework)*.

[ISO9075-2] ISO/IEC 9075-2:2011, *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 9075-11:2011, *Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata)*.

[ISO9075-13] ISO/IEC 9075-13:2008, *Information technology — Database languages — SQL — Part 13: SQL Routines and Types Using the Java™ Programming Language (SQL/JRT)*.

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™ Platform Standard Edition 6 API Specification*, `http://java.sun.com/javase/6/-docs/api/index.html`.

[Serialization] *Java™ Object Serialization Specification*, version 6.0 `http://java.sun.com/javase/-6/docs/platform/serialization/spec/serialTOC.html`.

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

(Blank page)

3 Routines tutorial

3.1 Technical components

Part 13 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 of OUT 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.

3.2 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.

3.3 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;
  }
}
```

3.4 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.

— 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.

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.

3.5 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))
  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.

3.6 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: