ISO/IEC JTC 1/SC 22/WG 23 N 0348

Proposed Annex for SQL Language

Date 2011-06-20

Contributed by Jim Johnson

Original file name SQL Annex.docx

Annex SQL

SQL. Vulnerability descriptions for the language SQL Standards and terminology

SQL.1 Identification of standards and associated documents

ISO 9075-1:2008 SQL/Framework
ISO 9075-2:2008 SQL/Foundation
ISO 9075-3:2008 SQL/Call-Level Interface
ISO 9075-4:2008 SQL/Persistent Stored Modules
ISO 9075-9:2008 SQL/Management of External Data
ISO 9075-10:2008 SQL/Object Language Bindings
ISO 9075-11:2008 SQL/Schemata
ISO 9075-13:2008 SQL/SQL Routines and Types Using the Java™ Programming Language
ISO 9075-2:2008 SQL/XML-Related Specifications

SQL.2 General Terminology and Concepts

***atomic*:** Incapable of being subdivided.
***compilation unit*:** A segment of executable code, possibly consisting of one or more subprograms.
***data type*:** A set of representable values.
***descriptor*:** A coded description of an SQL object. It includes all of the information about the object that a conforming SQL-implementation requires.
***identifier*:** A means by which something is identified.
***identify*:** To reference something without ambiguity.
***implementation-defined*:** Possibly differing between SQL-implementations, but specified by the implementor for each particular SQL-implementation.
***implementation-dependent*:** Possibly differing between SQL-implementations, but not specified by ISO/IEC 9075, and not required to be specified by the implementor for any particular SQL implementation.
***instance (of a value)*:** A physical representation of a value. Each instance is at exactly one site. An instance has a data type that is the data type of its value.
***null value*:** A special value that is used to indicate the absence of any data value.
***object (as in 'x object')*:** Any *thing*. An *X* object is a component of, or is otherwise associated with, some *X*, and cannot exist independently of that *X*. For example, an SQL object is an object that exists only in the context of SQL; an SQL-schema object is an object that exists in some SQL-schema.
***persistent*:** Continuing to exist indefinitely, until destroyed deliberately. Referential and cascaded actions are regarded as deliberate. Actions incidental to the termination of an SQL-transaction or an SQL-session are not regarded as deliberate.

***property (of an object)*:** An attribute, quality, or characteristic of the object.
***row*:** A sequence of (field name, value) pairs, the data type of each value being specified by the row type.
***scope (of a name or declaration)*:** The one or more BNF non-terminal symbols within which the name or declaration is effective.
***scope (of a reference type)*:** The table that a value of that reference type references.
***sequence*:** An ordered collection of objects that are not necessarily distinct.
***site*:** A place occupied by an instance of a value of some specified data type (or subtype of that data type).
***sizing item*:** sizing item: The value of an implementation-defined item for the SQL-implementation or for a profile.
***SQL-connection*:** An association between an SQL-client and an SQL-server.
***SQL-environment*:** The context in which SQL-data exists and SQL-statements are executed.
***SQL-implementation*:** A processor that processes SQL-statements. A *conforming SQL-implementation* is an SQL-implementation that satisfies the requirements for SQL-implementations as defined in ISO 9075-1 clause 8.
***SQL-session*:** The context within which a single user, from a single SQL-agent, executes a sequence of consecutive SQL-statements over a single SQL-connection.
***SQL-statement*:** A string of characters that conforms to the Format and Syntax Rules specified in the parts of ISO/IEC 9075.
***table*:** An unordered collection of rows having an ordered collection of one or more columns. Each column has a name and a data type. Each row has, for each column, exactly one value in the data type of that column.

SQL.3 Type System [IHN]

SQL.3.1 Applicability to language

SQL has a static type system. Each expression accepts a known set of operands and returns an expected type. If any types are not compatible with what is expected then an error is generated. Users can create custom types which are collections of base types.

SQL.3.2 Guidance to language users

- Be aware of compatibility between different types
- Know available types can differ between implantations

SQL.4 Bit Representations [STR]

This vulnerability does not apply to SQL.

SQL.5 Floating-point Arithmetic [PLF]

SQL.5.1 Applicability to language

SQL supports a numeric type known as approximate numeric, which is essentially floating point. The SQL data types FLOAT, REAL, and DOUBLE PRECISION are approximate numeric types. As with other

programming languages computations involving approximate values are prone to rounding and truncating errors.

SQL.5.2 Guidance to language users

- Understand the floating point format used to represent numbers.
- Do not use floating point to represent monetary values.

SQL.6 Enumerator Issues [CCB]

This vulnerability does not apply to SQL.

SQL.7 Numeric Conversion Errors [FLC]

SQL.7.1 Applicability to language

Implicit numeric conversion occurs in SQL when multiple numeric types are used. Unless used in procedural SQL, these conversions likely do not pose programmatic problems beside those already mentioned in SQL.5. Explicit casts are permitted in SQL using the CAST keyword. A complete table of acceptable casts can be found in SQL/Foundation 2008 Subclause 6.12.

SQL.7.2 Guidance to language users

- Realize situations which involve multiple numeric types
- Avoid situations which cause an exception because the final type cannot properly contain the value of an expression

---

SQL.8 String Termination [CJM]

This vulnerability does not apply to SQL.

SQL.9 Buffer Boundary Violation [HCB]

This vulnerability does not apply to SQL.

SQL.10 Unchecked Array Indexing [XYZ]

This vulnerability does not apply to SQL.

SQL.11 Unchecked Array Copying [XYW]

This vulnerability does not apply to SQL.

SQL.12 Pointer Casting and Pointer Type Changes [HFC]

This vulnerability does not apply to SQL.

SQL.13 Pointer Arithmetic [RVG]

This vulnerability does not apply to SQL.

SQL.14 Null Pointer Dereference [XYH]

This vulnerability does not apply to SQL.

SQL.15 Dangling Reference to Heap [XYK]

This vulnerability does not apply to SQL.

SQL.16 Wrap-around Error [XYY]

This vulnerability does not apply to SQL.

SQL.17 Using Shift Operations for Multiplication and Division [PIK]

This vulnerability does not apply to SQL.

SQL.18 Sign Extension Error [XZI]

This vulnerability does not apply to SQL.

SQL.19 Choice of Clear Names [NAI]

SQL.19.1 Applicability to language

The SQL standard provides naming mechanisms which aid in providing both readable and convenient names. In a query for example, fields from multiple tables can be referred to without qualifying their table so long as the field names are unique. In addition tables can be aliased in queries for cases where tables do need to be specified and for which their descriptive names would be too unwieldy to repeat multiple times.

SQL.19.2 Guidance to language users

- Users should follow a consistent convention
- Use names that are visually unambiguous
- Use names with rich meaning

SQL.20 Dead Store [WXQ]

SQL.20.1 Applicability to language

SQL allows assignments in both queries and procedural SQL. In both cases a user can assign a variable a value which is not subsequently used. In queries the effect is extra processing time, but most likely ill effects would be recognized immediately. Procedural SQL is more susceptible to this.

SQL.20.2 Guidance to language users

- Ensure each assignment is made to the intended variable
- Avoid unnecessary uses of assignment, especially in cases where clean code can be written instead

---

SQL.21 Unused Variable [YZS]

SQL.21.1 Applicability to language

Unused variables are possible in SQL. Distinction must be made in queries between unused variables and unused fields of a table. The difference being the former is a mistake and the latter is intention design of the language. All fields of a table may not be necessary, but their inclusion does no harm.

Procedural SQL extension gives more opportunity for this vulnerability to occur. Variables which are declared but not used should be considered errors.

SQL.21.2 Guidance to language users

- Ensure each variable is assigned a value

SQL.22 Identifier Name Reuse [YOW]

SQL.22.1 Applicability to language

The SQL language has many opportunities for identifier reuse. Tables can reuse names without worry about confusion, because each reused name must be explicitly prefixed with its table's name. There may be conflict when procedural SQL extension is added.

SQL.22.2 Guidance to language users

- Ensure a definition of an entity does not occur in a scope where a different entity with the same name is accessible

---

SQL.23 Namespace Issues [BJL]

This vulnerability does not apply to SQL.

SQL.24 Initialization of Variables [LAV]

This vulnerability does not apply to SQL. All variables are created with a default value.

SQL.25 Operator Precedence/Order of Evaluation [JCW]

SQL.25.1 Applicability to language

SQL establishes precedence for its operators. Users will mostly encounter these rules in math expressions or expressions using less common operators, e.g. bitwise operators. Are there other instances of operator precedence in SQL? Is there any guarantee of order of operation?

SQL.25.2 Guidance to language users

- Break up complex expressions and use temporary variables to make the order clearer.
- Use parentheses around binary operator combinations that are known to be a source of error.

SQL.26 Side-effects and Order of Evaluation [SAM]

SQL.26.1 Applicability to language
Are side effects a problem in SQL?

SQL.26.2 Guidance to language users

SQL.27 Likely Incorrect Expression [KOA]

SQL.27.1 Applicability to language

Likely incorrect expressions are possible in SQL, especially when using multiple WHERE clauses. When WHERE clauses contradict each other, the query's behavior is most likely not the user's intention. These situations should be recognizable to the user during coding, and certainly in testing.

SQL.27.2 Guidance to language users
- Ensure no WHERE clauses conflict

SQL.28 Dead and Deactivated Code [XYQ]

SQL.28.1 Applicability to language

Stored procedures which are not called could be indicative of coding error. Having code which is available yet not used is undesirable because it is potentially executable through an injection.

Dead code will never be executed due to an always true or always false branching instruction. Instances of this are not an error in itself but might indicate incorrect logic. If the user's intention was truly to include this code it should be documented.

SQL.28.2 Guidance to language users
- Remove dead code unless its presence serves a purpose.
- When the user identifies a conditional which always evaluates to the same value, this could indicate an earlier bug and additional testing may be needed.

SQL.29 Switch Statements and Static Analysis [CLL]

SQL.29.1 Applicability to language

SQL version of the switch statement is CASE. Undesirable results can occur when a user omits a specific case, therefore the evaluation of that condition becomes either the else clause (if supplied) or NULL. Another possibility for error is overlapping or repeated conditions. The user should know the order of evaluation of conditions is linear and the first one to evaluate to true is the one whose result is returned.

SQL.29.2 Guidance to language users

- Unless NULL is an acceptable value for conditions not specifically listed, provide an else clause.
- List conditions from most specific to least specific to minimize overlapping conditions.

SQL.30 Demarcation of Control Flow [EOJ]

SQL.30.1 Applicability to language

This may apply to procedural SQL. Do all control constructs end with an explicit terminator?

SQL.30.2 Guidance to language users

SQL.31 Loop Control Variables [TEX]

SQL.31.1 Applicability to language

Loop control variables can be changed from within a loop. This is not usual or even obvious to a reader of the source code.

SQL.31.2 Guidance to language users

- Avoid modifying the loop control variable in the body of its associated loop.
- In cases where such modification is necessary indicate this in documentation.

SQL.32 Off-by-one Error [XZH]

This vulnerability does not apply to SQL. Is this possible in procedural SQL?

SQL.33 Structured Programming [EWD]

SQL.33.1 Applicability to language

Control flow can make it easy or difficult for the user to comprehend source code. Ideally it will be well structured as easy to follow. Users should avoid including GOTOs as much as possible.

SQL.33.2 Guidance to language users

- Avoid using GOTOs as much as possible.
- Structure code into logical blocks that are easy for the reader.
-

SQL.34 Passing Parameters and Return Values [CSJ]

SQL.34.1 Applicability to language

SQL and procedural SQL both provide procedures which can be called with parameters. SQL appears to use call by copy, that is parameters appear as local variables to the procedures. This is evident by parameters values not being altered. When defining their own procedures users have more options.

Procedures can include three types of paramets: IN, OUT, and INOUT. As expected an IN is treated as a local variable and changes will not be visible outside of the procedure. Likewise modification is expected for OUT and INOUT parameters.

SQL.34.2 Guidance to language users

- Minimize side-effects and only include necessary OUT and INOUT parameters.
-

SQL.35 Dangling References to Stack Frames [DCM]

This vulnerability is not applicable to SQL. Please verify.

SQL.36 Subprogram Signature Mismatch [OTR]

SQL.36.1 Applicability to language

SQL checks the number and type of parameters passed to procedures. A mechanism exists for users to define default values for parameters, thereby making their inclusion in a call unnecessary. However, if an insufficient number of required parameters are not passed an exception will occur. This applies to built-in procedures as well.

How are external libraries handled?

SQL.36.2 Guidance to language users

- Review procedure calls where a mismatch may occur.
- Take advantage of any mechanisms provided by the implementation to ensure procedure signature matches.

SQL.37 Recursion [GDL]

SQL.37.1 Applicability to language

Procedural SQL permits the recursive invocation of procedures.

SQL.37.2 Guidance to language users

- Minimize the use of recursion.
- Convert recursive calculations to corresponding iterative calculations.
-

SQL.38 Ignored Error Status and Unhandled Exceptions [OYB]

SQL.38.1 Applicability to language

Procedural SQL has error handling which allows the user to catch exceptions in their programs. Does this same handling apply to SQL queries? What are the results of including no exception handling in code?

SQL.38.2 Guidance to language users

- Exceptions should be handled as close to the origin of the exception as possible.
- What is good advice for SQL error handling?

-

SQL.39 Termination Strategy [REU]

SQL.39.1 Applicability to language

An SQL environment is typically inside a database management system. In the event of a fault it is the system's responsibility to handle this as gracefully as it can. This could range from immediately halting the system to degraded performance, depending on the specifics of the SQL-environment.

SQL.39.2 Guidance to language users

- Are there any standard options SQL users can configure as to how an SQL-environment should handle faults?
-

SQL.40 Type-breaking Reinterpretation of Data [AMV]

This vulnerability does not apply to SQL.

SQL.41 Memory Leak [XYL]

This vulnerability is not applicable to SQL. Can SQL users cause memory leaks? This is not the same as the database management system leaking memory.

SQL.42 Templates and Generics [SYM]

This vulnerability is not applicable to SQL.

SQL.43 Inheritance [RIP]

SQL.43.1 Applicability to language

SQL has table inheritance, where one table can be based off an existing table. Does it support inheritance of custom data types?

SQL.43.2 Guidance to language users

-

SQL.44 Extra Intrinsics [LRM]

SQL.44.1 Applicability to language

SQL allows implementations to extend the language with procedures which are available to the user at all times.

SQL.44.2 Guidance to language users

- Be aware of the documentation for the specific implementation in use.
-

SQL.45 Argument Passing to Library Functions [TRJ]

SQL.45.1 Applicability to language

How does SQL handle passing arguments to library functions? Is it language specific, or is there a common interface?

SQL.45.2 Guidance to language users

- Libraries should be defined so that as many arguments as possible are validated.
- Use libraries known to have been developed with consistent and validated interface requirements.

SQL.46 Inter-language Calling [DJS]

SQL.46.1 Applicability to language

Applications developed in more than one language have challenges communicating between each language. Most often SQL is but one part of an entire software development. What conventions does SQL provide to allow calls to other languages? Likewise, is there a standard method for other languages interacting with SQL within the standard?

SQL.46.2 Guidance to language users

- Understand the calling conventions of all languages used.
-

SQL.47 Dynamically-linked Code and Self-modifying Code [NYY]

SQL.47.1 Terminology and features

Does SQL standard define interactions with dynamically-linked code? Can self-modifying code be created in SQL or procedural SQL?

SQL.47.2 Description of vulnerability

-

SQL.48 Library Signature [NSQ]

SQL.48.1 Terminology and features

Does SQL allow specification of library signatures? What about inter-language signatures?

SQL.48.2 Description of vulnerability

- Use tools to create signatures
-

SQL.49 Unanticipated Exceptions from Library Routines [HJW]

SQL.49.1 Terminology and features

What mechanisms in SQL are there to handle exceptions from libraries? What about libraries written in another language?

SQL.49.2 Description of vulnerability

- Wrap library calls within a catch-all exception handler.
- Or only use library routines for which all possible exceptions are specified.

SQL.50 Pre-processor Directives [NMP]

This vulnerability is not applicable to SQL.

SQL.51 Suppression of Language-defined Run-time Checking [MXB]

SQL.51.1 Terminology and features
Does the SQL standard allow suppression of run-time checks?

SQL.51.2 Description of vulnerability

-

SQL.52 Provision of Inherently Unsafe Operations [SKL]

SQL.52.1 Terminology and features
This might be applicable to SQL.

SQL.52.2 Description of vulnerability

-

SQL.53 Obscure Language Features [BRS]

SQL.53.1 Applicability of language
What features of SQL are obscure of misunderstood by many users?

SQL.53.2 Guidance to language users

-

SQL.54 Unspecified Behaviour [BQF]

SQL.54.1 Applicability of language

This vulnerability applies to SQL. What constructs does SQL provide which do not have specified behaviour.

SQL.54.2 Guidance to language users

-

SQL.55 Undefined Behaviour [EWF]

SQL.55.1 Applicability to language

What are the undefined behaviours of SQL?

SQL.55.2 Guidance to language users

- Ensure that undefined language constructs are not used.
-

SQL.56 Implementation –defined Behaviour [FAB]

SQL.56.1 Applicability to language

What SQL features are allowed to be implementation defined?

SQL.56.3 Guidance to language users

- Verify code behaviour in the intended SQL-environment or enivronments.
-

SQL.57 Deprecated Language Features [MEM]

SQL.57.1 Applicability to language

The SQL language has matured through several iterations of its standard. In this process features likely have been deprecated. What are examples of deprecated language features?

SQL.57.2 Guidance to language users

- Avoid the use of deprecated features
- Adhere to the latest published standard for which a suitable environment is available.
-
