SQL Coding Guidelines

Version 0.3

July 27, 2001

Table of Contents

1. Introduction 3

2 Problem areas when moving from one RDBMS to another 6

2.1 Minor issues 6

2.1.1 Semicolons 7

2.1.2 Continuation characters 7

2.1.3 The UNION keyword 7

2.1.4 Other differences in SQL variants/flavors 7

2.2 Important issues which are not too hard to resolve 7

2.2.1 Object naming - simple issues 7

2.2.2 Object naming - complicated issues 8

2.2.3 Types and type conversion 9

2.2.4 Dates and times 10

2.2.5 Joins 10

2.2.6 Scalar functions 11

2.2.7 Sophisticated techniques 11

2.3 Important issues which can be hard to resolve 13

Appendix A – Glossary 14

Appendix B – References 15

Appendix C – ISO-ANSI definition of a simple SELECT statement 16

Appendix D – Keywords/Reserved Words 19

Appendix E – Data Types 25

Appendix F – Date and Time 27

Appendix G – Scalar Functions 28

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Altered by** | **Date** | **Comment** |
| 0.1 | Leo Heska | July 23, 2001 | Draft |
| 0.3 | Leo Heska | July 27, 2001 | Draft |
|  |  |  |  |

# Introduction

It is not necessary to have and/or follow SQL guidelines.

SQL, being very English-language like, is fairly readable even if programmers follow no standards. Programmers who don't document what they're doing can write confusing SQL code, but a one-line standard, "use one good comment before each SQL statement" addresses this. If programmers use "clever tricks" or very sophisticated techniques, the code can be hard to understand. Another one-line standard, "explain your tricky/sophisticated SQL usage," can address this.

So using just two lines of SQL coding standards, you can create a SQL-based system that can run correctly, be understood by follow-on programmers, and be changed as necessary to meet future requirements. And that is all we can expect from any program, until and unless we want to change platforms.

But SQL guidelines can help us be "not married" to a particular RDBMS.

This document describes some programming guidelines for SQL application development. Guidelines used by DBAs are beyond the scope of this document. So are guidelines for testers who would ensure that an application still worked after moving it from one RDBMS to another.

* 1. **The need for these guidelines**

An application can outgrow the RDBMS it's running on when usage of the application increases to the point where the RDBMS can't handle the load. Or, applications might need to be moved to a different RDBMS because the old RDBMS is no longer supported, we stop licensing the old RDBMS, a corporate standardization program requires the move, new technologies work with a new RDBMS but not with the old one, or for other reasons.

The problem with moving from one RDBMS to another is that the existing SQL code may need to be rewritten to work on the new RDBMS. Depending on how the database was set up and how the code was written, the effort to rewrite the code can range as follows:

* Zero rewriting effort - the old SQL code runs fine on the new RDBMS.
* Unacceptably high rewriting effort - the old SQL code is so RDBMS-specific that it would take more effort to rewrite it than to just completely redesign and rewrite a new system.
* Somewhere in between the above two levels of effort.

A SQL guidelines document like this one can, if used, help minimize the effort required to move from one RDBMS to another.

* 1. **Why we don't just "follow the industry standard"**

**Reason 1: There is no one standard.**

SQL-89, SQL-92, and SQL-99 all exist, with SQL-92 and SQL-99 being most commonly referred and adhered to. Choosing between these standards is problematic; no one of them will do the job for us.

SQL-92 includes the concepts of "Entry", "Transitional", "Intermediate", and "Full" compliance. Most RDBMS vendors claiming compliance with "the SQL standard" claim SQL-92 entry-level compliance. But sticking to just entry-level SQL-92 is impractical; there is too much functionality which falls outside this subset. We need this "non SQL-92 entry-level" functionality to accomplish real-world SQL programming tasks.

Similarly, SQL-99 includes the concept of "Core" conformance level. RDBMS vendors state things like:

"[our product] is broadly compatible with the SQL-99 Core specification. However, some SQL-99 Core features are not currently implemented in [our product] or differ from [our] implementation."

So, sticking to SQL-99 Core doesn't accomplish our goal, either.

**Reason 2: Length of standards**

Reading/learning the standards is too much work for any programmer. The standards themselves are huge. SQL-92 is over 700 pages long; SQL-99 over 2000 pages.

**Reason 3: The standards are not written for SQL programmers**

The standards are written for RDBMS developers. They are not at all useful for programmers actually writing SQL. Plus, there are other problems. If you refer to "Appendix C – ISO-ANSI definition of a simple SELECT statement", you may note the following:

* It appears to imply that every SELECT statement must include an INTO clause, which is false.
* It doesn't even come close to showing all the things a programmer can do in a SELECT statement.
* It is too long.
* It requires the reader to understand BNF/EBNF.
* It is difficult to read, requiring hunting/tunneling around through the entire specification to find out what the various terms used in the definition mean.
  1. **Why we don't just write "JDBC SQL"**

Our group will be doing a lot of Java programming, and we'll be using JDBC drivers. So one way of addressing the "pick your standard" problem could be to follow the JDBC specification.

The JDBC specification requires, among other things, that implementations of JDBC drivers:

* Must support SQL92 plus the DROP TABLE statement.
* Must support "escape syntax", which translates from SQL92 to any variants the underlying RDBMS uses.
* Must support transactions (as specified in SQL99).
* Should support all features provided by the underlying RDBMS, even if that means extending the JDBC API. The idea is that using JDBC, a programmer can get the same functionality they'd get by accessing the RDBMS directly or via Call Level Interface.

Unfortunately, rather than simplifying things, adhering to the JDBC standard complicates them. It specifies using both SQL-92 and SQL-99, requires knowing those standards fairly well, and throws in the "wild card" that all underlying RDBMS functionality must be supported. This means that a programmer following the standard "make sure it runs using JDBC" could write RDBMS-specific code that runs fine, but fails when run against another RDBMS.

* 1. **Conclusion: our own in-house guidelines**

In conclusion, there is no industry standard which programmers can follow to ensure that the SQL they write will run against another RDBMS. If we want standards or guidelines, we'll have to come up with our own.

* 1. **What this document cannot do**

ISO and ANSI together have failed to create standards which programmers can follow to the letter, which will ensure that SQL they write will run against any "conformant" or "compliant" RDBMS. This document does not attempt to do what ISO and ANSI have failed to do after expending hundreds of person-years trying.

This document consists of guidelines only. Its goal is to help a programmer be aware of things they can do to increase the chances that their code will still work if ever run against/on another RDBMS. However, in some cases, it may be appropriate and desirable for a programmer to disregard these guidelines.

* 1. **How to use these guidelines**

Please do not attempt to use these guidelines as a "cookbook" for writing SQL code. As you will see, they are not detailed enough for that - they don't even contain a lot of SQL code.

**If you are a programmer**, this document may help you

* Write SQL code that is readable to other programmers, who are familiar with other dialects of SQL, and
* Write SQL code that will be relatively easy to move to a different RDBMS, if that is ever necessary.

The goal of these guidelines is to point out, remind you of, and/or help you focus attention on some of the common issues involved in trying to run SQL code written for one RDBMS to another. You, the programmer, will need to decide how to deal with these issues.

**If you are reviewing SQL code**, either alone or as part of a team, you will not in general be able to use these (or any other) guidelines as a way to enforce portable code. Restricting a SQL programmer to "standard" or "core" SQL means forgoing desirable, and, in some instances, required functionality.

However, this document may help you identify parts of the SQL code you're reviewing which could require rewriting in case of an RDBMS change. This could be the first step in figuring out a "level of effort" (how much work/how much money) required to convert the code.

* 1. **Author's note**

I have assembled this set of guidelines based on my own SQL programming experience and usage of RDBMS products. Therefore, this document contains omissions and, probably, errors. In some cases, I know an issue existed, but don't remember the particular RDBMSs or versions in which I encountered the issue. There may be important issues which belong here, but which I don't know about, because I happen to have never encountered them.

Also, any document like this one goes out of date. This document may identify issues which no longer exist because of changes to RDBMSs. For example, I know that Oracle 9i has made progress in supporting the ANSI/ISO standards, but I have not worked enough with Oracle 9i to know which issues still exist when moving to/from that product.

# Problem areas when moving from one RDBMS to another

In this document, I restrict my discussion to Oracle, Microsoft SQL Server, and IBM DB/2 or UDB, with occasional reference to the ANSI/ISO standard.

## Minor issues

By "minor", I mean differences which a human SQL programmer can easily identify and change to the "obvious equivalent" syntax in the other/destination SQL variant. Cleaning up these minor issues can seem like a simple or trivial task. Even so, they do come up and need to be dealt with. If we follow the rule "if you touch it, you break it", we recognize that even these trivial fixes require a decent regression test after the changes are made.

These issues all depend heavily on the SQL programming/execution environment. Many of these issues go away/do not apply to SQL which was written to run using JDBC.

### **Semicolons**

Oracle SQL statements sometimes need to end in a semicolon, whereas DB2 and SQL Server don't accept this.

### **Continuation characters**

Some SQL variants or programming environments require continuation characters for multi-line SQL statements.

### **The UNION keyword**

Some SQL variants or programming environments recognize the first complete SQL query, and object if they encounter a UNION keyword. I believe I encountered this when using ODBC.

### **Other differences in SQL variants/flavors**

There are actually tens, if not hundreds, of syntactic differences between SQL variants. However, many of these differences come from differences in function names, data types, et cetera, and are therefore not "minor". See sections below for a presentation of these issues.

## Important issues which are not too hard to resolve

### **Object naming - simple issues**

Different database implementations have different rules for naming database objects such as indexes, tables, and columns. Some restrictions are relatively easy to watch out for and conform to. These include name length, lexical restrictions, and "reserved words" or "keywords". I call these "simple naming issues".

#### Length of names of objects

Each RDBMS restricts the length of database object identifiers. Maximum lengths:

IBM UDB: 18 characters

Oracle: 30 characters

SQL Server: 128 characters

If you use more than 18 or 30 characters in your when naming your object, someone may need to rename them if the application ever needs to be moved to another RDBMS. Also, someone may need to resolve naming conflicts/confusion, if the new, shortened, names conflict and/or are identical.

#### Reserved words

In "Appendix D – Keywords/Reserved Words", I list reserved words for the three RDBMSs. It's a good idea to avoid using any of these words to name a database object.

Some RDBMSs allow you to use reserved words as object names by enclosing them in double quotes. This tends to degrade readability. For example:

SELECT "SELECT FROM"

FROM "WHERE"

WHERE "NULL" = 'STRING'

This could work if a table named "WHERE" has two columns named "SELECT FROM" and "NULL". You end up needing to use the quotation marks all the time, and this can be a hassle when using JDBC, but the code is, astoundingly, highly portable. Don't do this.

### **Object naming - complicated issues**

Some naming restrictions are not so easy to spot before actually attempting to get the code to run. For example, SQL Server allows identically named indexes, as long as each index is on a different table. Sybase, SQL Anywhere, and Oracle forbid this. For example, in SQL Server, the following works:

create table LeoJunk

(JunkNum numeric)

create table LeoJunk1

(JunkText varchar(10))

insert LeoJunk1 (JunkText) values ('trash')

-- OK, now we have two junk tables.

-- Attempt to create an identically-named index on each.

create index junkindex on leojunk(junknum)

create index junkindex on LeoJunk1(JunkText)

-- Works just fine.

But in Oracle, the following does not work:

create table LeoJunk

(JunkNum number);

create table leo\_junk

(JunkText varchar2(25));

create index junkindex on leojunk(junknum);

create index junkindex on leo\_junk(junktext);

The first of the two CREATE INDEX statements works, but the second is answered by the message "ORA-00955: name is already used by an existing object".

There are actually a lot of complicated naming issues like this, which you can't necessarily detect just by verifying that the SQL syntax is correct for a particular RDBMS. Only actually implementing the product, plus carrying out a good test, will catch all these errors.

### **Types and type conversion**

It's good to avoid type conversions wherever possible for performance reasons. It's also good to avoid them because if you're programming using JDBC, you already have the Java-to-RDBMS type conversion issue to deal with. And another reason to avoid type conversions is to increase portability between RDBMSs.

Except for very simple type conversions, each RDBMS does type conversion differently, and has its own set of scalar functions to perform the conversions. The methods and functions are not always compatible. We find ourselves faced with issues such as each RDBMS's internal data representation format, testing for roundoff errors and conversion accuracy, and figuring out just how each RDBMS does its conversion in a given context.

The ISO/ANSI standards specify that the CAST function should be used to convert between data types, for example:

SELECT field1, CAST(field2 AS INTEGER)

FROM table1

WHERE field1 = 'SMITH'

However, the various database vendors have different rules for which data types may be converted. You can attempt to get the ISO/ANSI standards, and then follow the casting rules defined therein. But, since most database vendors will implement the standard plus add their own functionality, if your application does any implicit or explicit casting and/or data type conversion, you should test very carefully when moving the application from one RDBMS to another. The test should consist at least of:

* a very careful pre-conversion code review, looking for type conversions
* a careful post-implementation side-by-side test of system behavior, designed to find irregularities and/or inequalities.

Values of the data types NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, and DOUBLE PRECISION may be compared and/or assigned to each other. If an assignment results in a loss of the most significant digits, the RDBMS should raise an exception, which you will notice. However, if least significant digits are lost, then implementation-defined rounding or truncating occurs with no exception being raised. You need to watch out for this, because different RDBMSs do their rounding/truncating differently.

You may assign values of types CHARACTER, VARCHAR, VARCHAR2, CLOB, and LOB directly to each other if they are taken from the same character set. If they are from different character sets, then a conversion to the destination character set must be performed if possible. If an assignment with truncation results in the loss of non-space characters, then the RDBMS is supposed to raise an exception.

Implicit type conversion can occur in expressions, fetch operations, single row select operations, inserts, deletes, and updates. You should always try to use CAST to make the conversion explicit.

Some data types do not exist in all database implementations. Examples include VARCHAR2 (Oracle) and BOOLEAN (ISO/ANSI standard type).

See "Appendix E – Data Types" for a list of supported data types.

### **Dates and times**

Dates and times are problematic when moving an application from one RDBMS to another. While most of the base types (DATE, TIME,TIMESTAMP) are supported, each RDBMSs:

* has its own way of interpreting date and time literal values
* has its own way of performing calculations on dates and times,
* handles year 2000 compliance and related issues differently, and
* provides its own set of date and time scalar functions and system registers (to extract the system time, deal with time zones, et cetera).

Also, the DATE type has been incorrectly implemented with respect to the ISO/ANSI standards. The ISO/ANSI standards say that a DATE value contains a YEAR, MONTH and DAY component only. However many implementations confuse DATE with TIMESTAMP or even make the two equivalent.

If your application uses dates, times, timestamps, et cetera, you should test very carefully when moving it from one RDBMS to another. The test should consist at least of:

a very careful pre-conversion code review, looking for usage of dates and times, and

a careful post-implementation side-by-side test of system behavior.

See "Appendix F – Date and Time" for a comparison of supported date/time features and functionality.

### **Joins**

RDBMSs vary greatly in the types of joins they allow and the syntax they support to accomplish the joins. The following join types are described in the ISO/ANSI standards:

* INNER (normal join)
* LEFT OUTER (include NULL columns from the left table)
* RIGHT OUTER (include NULL columns from the right table)
* FULL OUTER (include NULL columns from both of the joined tables)
* CROSS JOIN (Cartesian product)
* NATURAL … JOIN
* JOIN…ON
* JOIN…USING

The ISO/ANSI standards also describe the following "join-related" operations:

* INTERSECT
* UNION
* EXCEPT

The IBM and Microsoft join implementations are more or less aligned with the ISO/ANSI standards. The Oracle implementation uses non-standard syntax for outer joins. Oracle does not support the EXCEPT keyword but uses MINUS instead.

It may take a lot of work to convert from or to Oracle, if you do any of the above types of joins.

### **Scalar functions**

For a list of scalar functions and which database implementations support them, please see "Appendix G – Scalar Functions". A quick glance will show that there is little standardization across products. Therefore, if you ever need to migrate your SQL code, you can expect to spend quite a bit of time:

* examining your code to find scalar functions,
* figuring out how to recode the scalar functions you use,
* estimating the time to recode your scalar functions, and/or,
* recoding the scalar functions.

Do not underestimate the time it may take you to do this. A simple recoding of a CEIL or TRUNC function can have unexpected consequences which you may not find out without some fairly rigorous and well-designed testing. The problem is that everything can seem to work fine in "usual" conditions, but in boundary conditions (very small or large numbers, negative numbers, zero), subtle differences can emerge.

### **Sophisticated techniques**

There are a lot of sophisticted SQL programming techniques - way too many to even survey in this document. See Joe Celko's columns or book called "SQL For Smarties" for examples.

Generally, when using these techniques, you can and should expect to spend at least some time analyzing the application and figuring out if you expect the technique to work on the new RDBMS. Even if you do not need to recode, you can and should almost always expect to spend time carefully testing the application and/or the query/queries after the move.

Here are just a couple of examples of sophisticated programming techniques which can require recoding and/or retesting.

#### Updating multi-table views

I once heard that the SQL standard specifies that an UPDATE statement may only update one table at a time. Oracle, and I believe also Sybase, allow more than one table to be referenced by the UPDATE command. Oracle allows updates to multi-table views. This is a useful and relatively common programming technique. It's frequently used as a way to avoid writing a stored procedure.

If you are moving to a platform which does not support DML on multi-table views, you face a significant programming task, possibly requiring you write one or more stored procedures.

#### Using queries where attributes (columns) or relations (tables) are expected

Some RDBMSs, such as IBM and Sybase (and possibly SQL Server?) allow queries to appear where selected column expressions are normally expected. For example:

SELECT employee.persid, employee.firstname,

(

SELECT deptid

FROM staff WHERE staff.persid = employee.persid

),

employee.surname,

FROM employee

This syntax is not allowed by the ISO/ANSI standards, I have heard.

Oracle supports similar if not identical functionality. Anywhere any relation would normally occur, you can put a query there instead. If the "normally expected" relation could be sized M x N, and the query returns a result set sized M x N, Oracle will process the query without complaint. For example:

SELECT junknum, junktext,

(

SELECT junktext

FROM leo\_junk

WHERE junknum = 2

)

FROM leojunk1

works fine if the inner query returns a 1 x 1 result (a single value), because the inner query is in a place where one single value (a relation sized 1 x 1) would normally be expected and acceptable.

#### Selecting the first N rows of a resultset

Different RDBMSs support different ways of doing this. Here is a Microsoft example:

select top 10 \*

from subscriptions

order by SubDate

And here is a (similar) Oracle example which does the same kind of thing:

select \* from

(

select distinct policy\_nbr from life\_investor\_report

order by policy\_nbr desc

)

where rownum <= 4

And here is an equivalent "portable" or "cross-RDBMS" example:

select distinct policy\_nbr

from life\_investor\_report lir

where 4 >

(

select count(distinct policy\_nbr)

from life\_investor\_report

where policy\_nbr > lir.policy\_nbr

)

order by policy\_nbr desc

If you are writing this kind of query, expect to spend a lot of time recoding and testing your queries if you use the RDBMS-specific versions (which you very often must, for performance reasons). Even if you use "portable", "cross-RDBMS" code, you can find yourself spending time dealing with unexpected behaviors.[[1]](#footnote-1)\*

## Important issues which can be hard to resolve

If your application does any of the following, expect to spend a lot of time re-architecting, re-designing, recoding, and/or retesting when moving from one RDBMS to another:

* Changing the locking behavior.
* Performing distributed transactions.
* Using temporary tables.
* Making use of different character sets, sort order(s), and/or Unicode collations.
* Using OLAP features/services.
* Using "English-language" query features.
* Using features which permit data searches/manipulation based on "sub-atomic" data, such as text searches within a single column.
* Using features which enable running non-SQL routines from within the SQL statement. These include Oracle's PL/SQL and Java functionality and Microsoft's T-SQL EXECUTE.
* Doing DDL or DCL.
* Using stored procedures and/or functions.
* Using Sequences (in Oracle)
* Using Timestamp or Identity Columns (SQL Server).
* Using sophisticated character/string outputting functions, such as the formatting functions which Oracle supplies to transform dates into strings.
* Using SQL Server's ability to use pass-through queries to OLE DB data sources in place of relations(tables).
* Using Oracle's ROWID or ROWNUM pseudo-columns.
* Using PL/SQL, T-SQL, or any other procedural language in conjunction with SQL.
* Using user-defined aggregate functions.

Please note that the above is just a partial list.

# Appendix A – Glossary

**BNF** -Backus Normal Form or Backus-Naur form. A symbolic system used to define computer languages.

**DBA** - Database Administrator. The person who manages the RDBMS. Typical DBA responsibilities include but are not limited to installing the database (RDBMS) software, managing its interaction with hardware and the operating system, creating database instances, managing users/user accounts, granting permissions, monitoring and tuning the RDBMS, ensuring periodic backups, and working with developers, architects, and users to make sure that current and planned uses of the RDBMS succeed.

Data Definition Language.

The SQL sentences used to manipulate data within these objects are called DML's or Data Manipulation Language.

The SQL sentences, which are used to control the behavior of these objects, are called DCL's or Data Control Language.

**DCL** - Data Control Language. SQL statements which control access to database objects. The best (only?) examples are GRANT and REVOKE, which grant/revoke permission.

**DDL** - Data Definition Language. SQL statements which manipulate database objects. Examples include DROP TABLE and CREATE TABLE.

**DML** - Data Manipulation Language. SQL statements which can change information stored in the RDBMS. Specifically, the INSERT, UPDATE, and DELETE statements.

**EBNF** -Extended Backus-Naur Form. A symbolic system used to define computer languages; an extension and refinement of BNF.

**RDB** - Relational Database. Data stored in a way consistent with the scheme invented by Codd (see Codd 1970).

**RDBMS** - Relational Database Management System. Generally, the hardware (including network and computer) and software (including operating system and database software) which together provide the ability to use (store, retrieve, change) data in a RDB. The term RDBMS is often used to refer just to the database software itself, such as Oracle 9I or Microsoft SQL Server 2000.

**SQL** - Structured Query Language. The language most commonly used to manipulate data in an RDBMS.

**UDB** - Universal Database. IBM's name for their database product. As far as I can tell, UDB means "DB/2 which can run on many platforms".

# Appendix B – References

ANSI X3.135-1986, "Database Language SQL". This is SQL-86.

ANSI X3.135-1989, "Database Language SQL". This is SQL-89, sometimes known as SQL-1.

ANSI X3.135-1992, "Database Language SQL". This is SQL-92, sometimes known as SQL-2. It's over 700 pages long.

ANSI X3.135-1999, "Database Language SQL". See ISO/IEC 9075:1999 below.

E. F. Codd. A relational model of data for large shared data banks. Communications of the ACM, 13(6):377--387, June 1970. <http://www.acm.org/classics/nov95/toc.html>

ISO/IEC 9075:1999, "Database Language SQL". This is SQL-99, sometimes known as SQL-3, and is the same as ANSI X3.135-1999. The standard consists of many parts, of which the first 5 are:

SQL/Framework - an introduction - 75 pages

SQL/Foundation - "core" SQL - 1100 pages

SQL/Call-level interface (CLI) - related to ODBC and JDBC - 400 pages

SQL/Persistent Stored Modules (PSM) - related to stored procedures - 160 pages

SQL/Bindings - embedded and dynamic SQL - 250 pages

# Appendix C – ISO-ANSI definition of a simple SELECT statement

Here is the definition of a single-row SELECT statement taken from one of the ISO-ANSI specification documents:

<select statement: single row>

Function

Retrieve values from a specified row of a table.

Format

<select statement: single row> ::=

SELECT [ <set quantifier> ] <select list>

INTO <select target list>

<table expression>

<select target list> ::=

<target specification> [ { <comma> <target specification> }... ]

Syntax Rules

Case:

If the <select target list> contains a single <target

specification> TS and the number of elements in the <select

target list> is greater than 1, then:

The data type of TS shall be a row type.

The Syntax Rules of Subclause 9.1, "Retrieval assignment",

apply to TS and a row type whose i-th element data type is

the data type of the i-th element of the <select list>, as

TARGET and VALUE, respectively.

Otherwise:

The number of elements in the <select list> shall be the

same as the number of elements in the <select target list>.

The i-th <target specification> in the <select target list>

corresponds with the i-th element of the <select list>.

The Syntax Rules of Subclause 9.1, "Retrieval assignment",

apply to each corresponding <target specification> and

<value expression>, as TARGET and VALUE, respectively.

Let S be a <query specification> whose <select list> and <table

expression> are those specified in the <select statement: single

row> and that specifies the <set quantifier> if it is specified

in the <select statement: single row>. S shall be a valid <query

specification>.

Access Rules

None.

General Rules

Let Q be the result of <query specification> S.

Case:

If the cardinality of Q is greater than 1, then an

exception condition is raised: cardinality violation. It

is implementation-dependent whether or not SQL-data values

are assigned to the targets identified by the <select target

list>.

If Q is empty, then no SQL-data values are assigned to

any targets identified by the <select target list>, and a

completion condition is raised: no data.

Otherwise, values in the row of Q are assigned to their

corresponding targets.

If a completion condition no data has been raised, then no

further General Rules of this Subclause are applied.

Case:

If the <select target list> contains a single <target

specification> TS and the number of elements in the <select

list> is greater than 1, then the current row is assigned

to TS and the General Rules of Subclause 9.1, "Retrieval

assignment", are applied to TS and a row whose i-th element

is the value of the i-th element of the <select list>, as

TARGET and VALUE respectively.

Otherwise:

The assignment of values to targets in the <select target

list> is in an implementation-dependent order.

The target identified by the i-th <target specification> of

the <select target list> corresponds to the i-th value in

the row of Q.

Let TV be an identified target and let SV be its

corresponding value in the row of Q.

The General Rules of Subclause 9.1, "Retrieval

assignment", are applied to TV and SV, as TARGET and VALUE,

respectively.

If an exception condition is raised during the assignment

of a value to a target, then the values of all targets are

implementation-dependent.

Leveling Rules

The following restrictions apply for Full SQL:

None.

The following restrictions apply for Intermediate SQL in

addition to any Full SQL restrictions:

None.

The following restrictions apply for Entry SQL in addition to

any Intermediate SQL restrictions:

If the data type of the target identified by the i-th <target

specification> in the <select target list> is an exact

numeric type, then the data type of the i-th column of the

table T shall be an exact numeric type.

The <table expression> shall not include a <group by clause>

or a <having clause> and shall not identify a grouped view.

# Appendix D – Keywords/Reserved Words

Keyword/Reserved Word IBM Oracle MS SQL Server

ACQUIRE X

ADD X X X

ALL X X X

ALTER X X X

AND X X X

ANY X X X

AS X X X

ASC X X X

AUDIT X

AUTHORIZATION X

AVG X

BACKUP X

BEGIN X

BETWEEN X X X

BIGINT X X

BINARY X X

BIT X X

BOTTOM X

BREAK X

BUFFERPOOL X

BY X X X

CALL X X

CAPTURE X

CASCADE X

CASE X X

CAST X X

CCSID X

CHAR X X X

CHAR\_CONVERT X

CHAR\_LENGTH

CHARACTER X X

CHECK X X X

CHECKPOINT X

CLOSE X

CLUSTER X

COLLECTION X

COLUMN X X

COMMENT X X X

COMMIT X X

CONCAT X

CONNECT X X X

CONNECTION X

CONSTRAINT X X

CONTINUE X

CONVERT X

COUNT X

CREATE X X X

CROSS X X

CURRENT X X X

CURRENT\_DATE X

CURRENT\_SERVER X

CURRENT\_TIME X

CURRENT\_TIMESTAMP X

CURRENT\_TIMEZONE X

CURRENT\_USER X

CURSOR X X

DATABASE X X

DATE X X X

DAY X

DAYS X

DBA X

DBSPACE X X

DEALLOCATE X

DEC X

DECIMAL X X

DECLARE X

DEFAULT X X X

DELETE X X X

DESC X X X

DESCRIPTOR X

DISABLE X

DISTINCT X X X

DO X

DOUBLE X X

DROP X X X

DYNAMIC X

EDITPROC X

ELSE X X X

ELSEIF X

ENABLE X

ENCRYPTED X

END X

END-EXEC X

ENDIF X

ERASE X

ESCAPE X X

EXCEPT X

EXCEPTION X X

EXCLUSIVE X

EXEC X

EXECUTE X X

EXISTING X

EXISTS X X X

EXPLAIN X

EXTERNAL X

EXTERNLOGIN X

FETCH X X

FIELDPROC X

FIRST X

FLOAT X X

FOR X X X

FOREIGN X X

FORWARD X

FROM X X X

FULL X X

GO X

GOTO X X

GRANT X X X

GRAPHIC X

GROUP X X X

HAVING X X X

HOLDLOCK X

HOUR X

HOURS X

IDENTIFIED X X

IF X

IMMEDIATE X

IN X X X

INCREMENT X

INDEX X X X

INDICATOR X

INNER X X

INOUT X X

INSENSITIVE X

INSERT X X X

INSTALL X

INSTEAD X

INT X

INTEGER X X

INTEGRATED X

INTERSECT X

INTO X X X

IQ X

IS X X X

ISOLATION X X

JOIN X X

KEY X X

LABEL X

LEFT X X

LEVEL X

LIKE X X

LOCK X X

LOCKSIZE X

LOGIN X

LONG X X X

MATCH X

MAX X

MAXVALUE X

MEMBERSHIP X

MESSAGE X

MICROSECOND X

MICROSECONDS X

MIN X

MINUS X

MINUTE X

MINUTES X

MINVALUE X

MODE X X X

MODIFY X X

MONTH X

MONTHS X

NAMED X

NATURAL X

NEW X

NHEADER X

NO X

NOHOLDLOCK X

NOT X X X

NOTIFY X

NULL X X X

NUMBER X

NUMERIC X X

NUMPARTS X

OBID X

OF X X X

OFF X

ON X X X

ONLY X

OPEN X

OPTIMIZE X

OPTION X X

OPTIONS X

OR X X X

ORDER X X X

OTHERS X

OUT X X

OUTER X X

PACKAGE X

PAGE X

PAGES X

PART X

PASSTHROUGH X

PCTFREE X X

PCTINDEX X

PLAN X

PRECISION X X

PREPARE X

PRIMARY X X

PRINT X

PRIOR X

PRIQTY X

PRIVATE X

PRIVILEGES X X X

PROC X

PROCEDURE X X

PROGRAM X

PUBLIC X X

PUBLICATION X

QUARTER X

RAISERROR X

RAW X

READTEXT X

REAL X X

REFERENCE X

REFERENCES X X

RELEASE X X

REMOTE X

REMOVE X

RENAME X

RESET X

RESOURCE X X

RESTORE X

RESTRICT X

RETURN X

REVOKE X X X

RIGHT X X

ROLLBACK X X

ROW X X

ROWID X

ROWS X X

RRN X

RUN X

SAVE X

SAVEPOINT X

SCHEDULE X X

SCHEMA X

SCROLL X

SECOND X

SECONDS X

SECQTY X

SELECT X X X

SEQUENCE X

SESSION X

SET X X X

SETUSER X

SHARE X X

SIMPLE X

SIZE X

SMALLINT X X

SOME X X

SQLCODE X

SQLSTATE X

START X X

STATISTICS X

STOGROUP X

STOP X

STORPOOL X

SUBCLASS\_ORIGIN

SUBPAGES X

SUBSTRING X

SUBTRANS X

SUBTRANSACTION X

SUCCESSFUL X

SUM X

SYNCHRONIZE X

SYNONYM X X

SYNTAX\_ERROR X

SYSDATE X

TABLE X X X

TABLESPACE X

TEMPORARY X

THEN X X

TIME X X

TIMESTAMP X X

TINYINT X X

TO X X X

TOP X

TRAN X

TRANSACTION X

TRIGGER X

TRIM X

TRUNCATE X

TSEQUAL X

UID X

UNION X X X

UNIQUE X X X

UNKNOWN X

UNSIGNED X

UPDATE X X X

USER X X X

USING X X

VALIDATE X X

VALIDPROC X

VALUES X X X

VARBINARY X X

VARCHAR X X

VARCHAR2 X

VARIABLE X X

VARYING X

VCAT X

VIEW X X X

VOLUMES X

WHEN X

WHENEVER X

WHERE X X X

WHILE X

WITH X X X

WORK X X

YEAR X

YEARS X

# Appendix E – Data Types

X = supported

2.0 = supported in version 2.0 only (not in version 1.2)

Data Type IBM Oracle MS ISO ODBC JDBC

BIGINT X X X X X

BINARY X X X X

BINARY LARGE OBJECT X

BIT X X X X

BIT VARYING X

BLOB X 2.0

CHAR X X X X X X

CHAR VARYING X

CHARACTER X X

CHARACTER LARGE OBJECT X

CHARACTER VARYING X X

CLOB X 2.0

DATALINK X

DATE X X X X X X

DATETIME X

DBCLOB X

DEC X

DECIMAL X X X X X X

DOUBLE X X X

DOUBLE PRECISION X X X

ENUMERATED X

FLOAT X X X X X X

GRAPHIC X

IMAGE X

INT X X

INTEGER X X X X X

INTERVAL X

LOB X

LONG X

LONG BINARY X

LONG RAW X

LONG VARBINARY X X X

LONG VARCHAR X X X X X

LONG VARGRAPHIC X

MONEY X

NUMBER X

NUMERIC X X X X X X

RAW X

REAL X X X X X X

ROWID X

SMALLDATETIME X

SMALLINT X X X X X X

SMALLMONEY X

TEXT X

TIME X X X X X X

TIMESTAMP X X X X X X

TINYINT X X X X

UNSIGNED INT X

UNSIGNED SMALLINT X

UNSIGNED TINYINT X

VARBINARY X X X X

VARCHAR X X X X X X

VARCHAR2 X

VARGRAPHIC X

# Appendix F – Date and Time

Feature/Functionality IBM Oracle MS ISO

ADD\_MONTHS function X

CURDATE function X

CURTIME function X

CURRENT { DATE TIME TIMESTAMP } X X

CURRENT\_DATE X X

CURRENT\_TIME X X

CURRENT\_TIMESTAMP X X

DATE function X X

DATE type includes timestamp X X

DATEADD function X

DATEDIFF function X

DATEFORMAT X

DATENAME function X

DATEPART function X

DATETIME function X

DAY function X X

DAYNAME function X X

DAYOFWEEK function X X

DAYOFYEAR function X X

DAYS function X X

DOW function X

EXTRACT(field FROM datetime) X

GETDATE function X

HOUR function X X

HOURS function X

INTERVAL X

JULIAN\_DAY function X

LAST\_DAY X

MICROSECOND function X

MIDNIGHT\_SECONDS X

MINUTE function X X

MINUTES function X

MONTH function X X

MONTHNAME function X X

MONTHS function X

MONTHS\_BETWEEN function X

NEXT\_DAY function X

NOW function X

OVERLAPS X

QUARTER function X X X

ROUND function X

SECOND function X X

SECONDS function X

SYSDATE X

TIME function X

TIMESTAMP function X

TIMESTAMP\_ISO function X

TIMESTAMPADD function X

TIMESTAMPDIFF function X X

TO\_CHAR function X

TO\_DATE function X

TO\_NUMBER function X

TODAY function X

TRUNC function X

WEEK function X X X

WEEKS function X

YEAR function X X

YEARS function X

YMD function X

# Appendix G – Scalar Functions

Function IBM Oracle MS ISO

|| X X X X

ABS X X

ABSVAL X

ACOS X X

ADD\_MONTHS X

ARGN X

ASCII X X X

ASIN X X

ATAN X X

ATAN2 X X

AVG X X X X

BIGINT X

BIT\_LENGTH X X

BLOB X

BYTE\_LENGTH X

BYTE\_SUBSTR X

CASE X X

CAST X X X

CEIL X X

CEILING X

CHAR X X

CHAR\_LENGTH X X X

CHARACTER\_LENGTH X

CHARINDEX X

CHR X X

CLOB X

COALESCE X X

COLLATE X

CONCAT X X

CONVERT X X

COS X X

COT X X

COUNT X X X X

COUNT\_BIG X

COUNT\_STDDEV X

CURRENT\_DATE X

CURRENT\_TIME X

CURRENT\_TIMESTAMP X

CURRENT\_USER X

DATABASE X

DATE X X

DATEADD X

DATEDIFF X

DATEFORMAT X

DATENAME X

DATEPART X

DATETIME X

DAY X X

DAYNAME X X

DAYOFWEEK X X

DAYOFYEAR X X

DAYS X X

DBCLOB X

DEC X

DECIMAL X

DEGREES X X

DEREF X

DIFFERENCE X X

DIGITS X

DLCOMMENT X

DLLINKTYPE X

DLURLCOMPLETE X

DLURLPATH X

DLURLPATHONLY X

DLURLSCHEME X

DLURLSERVER X

DLVALUE X

DOUBLE X

DOUBLE\_PRECISION X

DOW X

ESTIMATE X

ESTIMATE\_SOURCE X

EVENT\_MON\_STATE X

EXP X X

EXPERIENCE\_ESTIMATE X

EXTRACT X

FLOAT X

FLOOR X X X

GENERATE\_UNIQUE X

GETDATE X

GRAPHIC X

GREATEST X

GROUPING X

HEX X

HEXTOINT X

HOUR X

HOURS X

IFNULL X X

INDEX\_ESTIMATE X

INITCAP X

INSERT X

INSERTSTR X

INSTR X

INSTRB X

INT X

INTEGER X

INTTOHEX X

ISNULL X

JULIAN\_DAY X

LAST\_DAY X

LCASE X X X

LEAST X

LEFT X X

LENGTH X X X

LENGTHB X

LIST X

LN X

LOCATE X X

LOG X X

LOG10 X X

LONG\_VARCHAR X

LONG\_VARGRAPHIC X

LOWER X X

LPAD X

LTRIM X X X

MAX X X X X

MICROSECOND X

MIDNIGHT\_SECONDS X

MIN X X X X

MINUTE X X

MINUTES X

MOD X X

MONTH X X

MONTHNAME X X

MONTHS X

MONTHS\_BETWEEN X

NEXT\_DAY X

NODENUMBER X

NOW X

NULLIF X

NUMBER X

NVL X

OCTET\_LENGTH X X

PARTITION X

PATINDEX X

PI X

PLAN X

POSITION X X

POSSTR X

POWER X X

QUARTER X X X

RADIANS X X

RAND X X

REAL X

REMAINDER X

REPEAT X X

REPLACE X X

RIGHT X X

ROUND X X X

RPAD X

RTRIM X X X

SECOND X X

SECONDS X

SESSION\_USER X

SIGN X X

SIMILAR X

SIN X X

SMALLINT X

SOUNDEX X X

SPACE X X

STR X

STRING X

SQRT X X

STDDEV X

STUFF X

SUBSTR X X X

SUBSTRB X

SUBSTRING X

SUM X X X X

SYSDATE X

SYSTEM\_USER X

TABLE\_NAME X

TABLE\_SCHEMA X

TAN X X

TIME X

TIMESTAMP X

TIMESTAMPADD X

TIMESTAMP\_ISO X

TIMESTAMPDIFF X X

TO\_CHAR X

TO\_DATE X

TO\_NUMBER X

TODAY X

TRACEBACK X

TRANSLATE X X X

TRIM X X X

TRUNC X X

TRUNCATE X

TYPE\_ID X

TYPE\_NAME X

TYPE\_SCHEMA X

UCASE X X X

UPPER X X

USER X

VALUE X

VARCHAR X

VARGRAPHIC X

VAR X

VARIANCE X X

WEEK X X

WEEKS X

YEAR X X

YEARS X

YMD X

1. \* For example, at first glance, you might expect that you could eliminate the DESC keyword from the Oracle and "portable" versions of the query. But doing so causes the two queries to return two entirely different result sets; not with different, but different content. [↑](#footnote-ref-1)