A Guide to the SQL Standard

C. J. Date

Preface

The purpose of this book is to describe the relational database language SQL. SQL has been adopted as an official standard by the American National Standards Institute (ANSI), and at the time of writing (late 1986) it looks very likely that it will soon also be adopted as an international standard by the International Standards Organization (ISO). In addition, numerous SQL-based products are already available in the marketplace (over 50 at the latest count).

As some readers may be aware, I have already discussed the SQL language at considerable length in several of my other books—including in particular *A Guide to DB2* (Addison-Wesley, 1984) and *A Guide to INGRES* (Addison-Wesley, 1987)—and I am very conscious that I may be accused of writing the same book over and over again "until I get it right." However, the treatment of SQL in the present book differs from that in those earlier books in a number of significant ways:

- The emphasis is on the official standard version of SQL instead of on one of the implemented dialects. The book should thus be relevant to *anyone* interested in the SQL language and SQL implementations—not just those from "the IBM world," which is where SQL originated, but also those with an interest in SQL implementations for, e.g., DEC, Data General, Honeywell, ICL, . . . and other environments.
- The emphasis in the standard on the use of SQL for *programmed* (as opposed to interactive) access to the database has many ramifications and repercussions on the way the book is structured and the way the material is presented. In some ways the discussions are almost the reverse of what they were in the earlier books; those earlier presentations

concentrated primarily on interactive SQL and discussed programming SQL at the end, almost as an afterthought. The present book, by contrast, necessarily deals almost exclusively with the use of SQL by programs.

- The treatment is more thorough. All aspects of the language are discussed in detail. In the earlier books, by contrast, I was not aiming at any such completeness, and it was expedient to simplify and/or ignore certain aspects of the language.
- At the same time, the book is (I hope) more "user-friendly" than the official SQL standard, in that it includes a more tutorial treatment of the material. The official standard is not very easy to read—partly because it necessarily reflects the structure of the SQL language itself, which in some ways is very ill-structured (despite the fact that the "S" in SQL stands for "Structured"!), and partly also because it tends to present the language bottom up instead of top down.
- It follows from the previous two paragraphs that the book is intended both as a work of reference and as a tutorial guide; it includes both formal definitions and numerous worked examples. However, I must make it very clear that the book is not intended to replace the official standard document but to complement it.
- The book also includes a discussion of the differences between the standard version of SQL and the IBM version supported by the IBM product DB2, an indepth examination of the official proposed extensions to the standard for functions such as referential integrity, an annotated SQL critique, an annotated bibliography, and other relevant items.
- Finally, the book also includes numerous comments on (and criticisms of) the standard. Such matter is set off from the body of the text by "Comment" and "End of comment" delimiters, in order to be readily distinguishable.

The book consists of twelve chapters and a set of appendices. The twelve chapters fall into three broad groups, as follows:

1. Introductory (Chapters 1–3)
2. The standard in detail (Chapters 4–10)
3. Possible extensions (Chapters 11–12)

Most of the examples are based on the familiar suppliers-and-parts database (see Chapter 2). I make no apology for using this old warhorse still one more time; basing the examples on such a familiar database should (I hope) make it easy for the reader to relate to those examples, and should

also facilitate comparisons between the standard version of SQL and specific vendor implementations—in particular, the implementations described in *A Guide to DB2* and *A Guide to INGRES*. In some respects, in fact, the book can be seen as a complement to those latter two books.

The book is intended to be reasonably self-contained. The only background assumed of the reader is a general interest in the SQL language. All relevant terms and concepts are defined and explained as they are introduced.

ACKNOWLEDGMENTS

As usual, I am delighted to acknowledge my debt to the many people involved, directly or indirectly, in the production of this book. First, it is a pleasure to acknowledge my gratitude to Phil Shaw, the IBM representative to the ANSI Database Committee, for his patience and assistance with my numerous technical questions. Second, I am pleased to be able to thank my reviewers Randell Flint, Carol Joyce, Geoff Sharman, and Phil Shaw (again) for their many helpful comments on the manuscript. Third, I am deeply indebted to my long-suffering family and to numerous friends and colleagues for their support throughout this project. Last, I am (as always) grateful to my editor, Elydia Siegel, and to the staff at Addison-Wesley for their assistance and their continually high standards of professionalism. It has been (as always) a pleasure to work with them.

Saratoga, California C. J. Date

It is only fitting to dedicate this book to
the many people responsible, directly or indirectly,
for the rise of SQL to its present preeminent position—
the original SQL language designers in IBM,
the implementers of the IBM prototype System R
and the IBM products (DB2 and SQL/DS)
derived from that prototype,
and the ANSI Database Committee X3H2.
I hope this book does justice to their efforts.

Contents

1

Why SQL Is Important

1.1 BACKGROUND

The name "SQL"—the official pronunciation is "ess-cue-ell," but most people usually pronounce it "sequel"—was originally an acronym, standing for "Structured Query Language." The SQL language consists of a set of facilities for defining, manipulating, and controlling data in a relational database. In order to understand why the language has become so widespread and so generally important, it is helpful to have an appreciation of some of the major developments in database technology over the past fifteen or so years. We therefore begin by summarizing those developments.

1. In 1970, E. F. Codd, at that time a member of the IBM Research Laboratory in San Jose, California, published a now classic paper, "A Relational Model of Data for Large Shared Data Banks" (*Communications of the ACM*, Vol. 13, No. 6, June 1970), in which he laid down a set of abstract principles for database management: the so-called *relational model*. The entire field of relational database technology has its origins in that paper. Codd's ideas led directly to a great deal of experimentation and research in universities, industrial research laboratories, and similar establishments, and that activity in turn led to the numerous relational products now available in the marketplace. The many advantages of the relational approach are far too well known to need repeating here; see, e.g., either

of the author's books *A Guide to DB2* (Addison-Wesley, 1984) or *A Guide to INGRES* (Addison-Wesley, 1987) for a discussion of those advantages.

2. One particular aspect of the research just referred to was the design and prototype implementation of a variety of relational languages. A relational language is a language that realizes, in some concrete syntactic form, some or all of the features of the abstract relational model. Several such languages were created in the early and mid 1970s. One such language in particular was the "Structured English Query Language" (SEQUEL), defined by D. D. Chamberlin and others at the IBM San Jose Research Laboratory (1974) and first implemented in an IBM prototype called SEQUEL-XRM (1974–75).

3. Partly as a result of experience with SEQUEL-XRM, a revised version of SEQUEL called SEQUEL/2 was defined in 1976–77. (The name was subsequently changed to SQL for legal reasons.) Work began on another, more ambitious, IBM prototype called System R. System R, an implementation of a large subset of the SEQUEL/2 (or SQL) language, became operational in 1977 and was subsequently installed in a number of user sites, both internal IBM sites and also (under a set of joint study agreements) selected IBM customer sites. *Note*: A number of further changes were made to the SQL language during the lifetime of the System R project, partly in response to user suggestions; for instance, an EXISTS function was added to test whether some specified data existed in the database.

4. Thanks in large part to the success of System R, it became apparent in the late 1970s that IBM would probably develop one or more products based on the System R technology—specifically, products that implemented the SQL language. As a result, other vendors also began to construct their own SQL-based products. In fact, at least one such product, namely ORACLE, from Relational Software Inc. (subsequently renamed Oracle Corporation), was actually introduced to the market prior to IBM's own products. Then, in 1981, IBM did announce a SQL product, namely SQL/DS, for the DOS/VSE environment. IBM then followed that announcement with one for a VM/CMS version of SQL/DS (1982), and another for an MVS product called DB2 that was broadly compatible with SQL/DS (1983).

5. Over the next several years, numerous other vendors also announced SQL-based products. Those announcements included both entirely new products such as DG/SQL (Data General Corporation, 1984) and SYBASE (Sybase Inc., 1986), and SQL interfaces to established products such as INGRES (Relational Technology Inc., 1981, 1985) and the IDM (Britton-Lee Inc., 1982, 1985). There are now (1986) some fifty or so products in the marketplace that support some dialect of SQL, running on machines

that range all the way from quite small micros to the largest mainframes. *SQL has become the de facto standard in the relational database world.*

6. SQL has also become an *official* standard. In 1982, the American National Standards Institute (ANSI) chartered its Database Committee (X3H2) to develop a proposal for a standard relational language. The X3H2 proposal, which was finally ratified by ANSI in 1986, consisted essentially of the IBM dialect of SQL, "warts and all" (except that a few—in this writer's opinion, far too few—minor IBM idiosyncrasies were removed). At the time of writing, it looks likely that the X3H2 proposal will soon be accepted as an international standard by the International Standards Organization (ISO).

From this point on we will generally take the unqualified name "SQL" to refer to the official standard version of the language. From time to time we may also use the term "standard SQL" for emphasis. We will always use qualified names such as "DB2 SQL" to refer to specific implemented dialects.

In many ways, the SQL standard is not particularly useful in itself; it has been characterized, perhaps a little unkindly, as "the intersection of existing implementations,"* and as such is severely deficient in a number of respects (see Section 1.2). Recognizing this fact, the X3H2 Committee is currently at work on a set of proposed extensions to the base standard. We will discuss some of those extensions in this book—but the reader is cautioned that they *are* only proposed extensions and are not (yet) part of the official standard, and hence are subject to possible change.

Following on from the previous point: It is only fair to stress that, while (as already stated) there are some fifty or so SQL implementations available today, no two of those implementations are precisely identical, and none of them is precisely identical to standard SQL! Even the IBM implementations in SQL/DS and DB2 are not 100 percent compatible with each other, and each of them differs from System R SQL and also from standard SQL on numerous points of detail—not all of them trivial, incidentally. This state of affairs may possibly change with time, of course. In this book we will be concentrating on standard SQL specifically (for the most part); however, we will also mention certain major deviations from the standard. In

*More accurately, this description refers to "Level 1" of the standard (see Appendix C). But the comment does highlight a general criticism, which is that the SQL standard, at least in its first version, seems more concerned with protecting existing vendor implementations than with establishing a truly solid foundation for the future.

particular, we will indicate where IBM (in the shape of DB2, and possibly SQL/DS) supports some significant feature that is not part of standard SQL.

One final point of a historical nature: The original version of SQL was intended for standalone, interactive use. However, facilities were added later to allow the invocation of SQL operations from an application programming language such as COBOL or PL/I. By contrast, the SQL standard concentrates almost exclusively on those latter (application programming) facilities, presumably on the grounds that standardization is much more significant for portability of programs than it is for interactive interfaces. This emphasis is reflected in the structure of the present book, as will be seen.

1.2 IS A STANDARD DESIRABLE?

Before going any further, we should perhaps consider the question of whether a SQL-based standard is a good thing. On the one hand, the advantages are fairly obvious:

- *Reduced training costs*: Application developers can move from one environment to another without the need for expensive retraining.
- *Application portability*: Applications—in particular, applications developed by third-party software vendors—can run unchanged in a variety of different hardware and software environments. Applications can be developed in one environment (e.g., on a PC) and then run in another (e.g., on a large mainframe).*
- *Application longevity*: Standard languages are assured of a reasonably long lifetime. Applications developed using such languages are therefore assured of a reasonably long lifetime also.
- *Cross-system communication*: Different systems can more easily communicate with one another. In particular, different database management systems might be able to function as equal partners in a single distributed database system if they all support the same standard interface.

---

*A note of caution is in order here. SQL is a *database* language, not a complete programming language; a typical application will involve, not only SQL statements, but also statements in some host language such as COBOL. The portability of such an application will thus depend on the portability of the host language as well as on that of SQL.

- *Customer choice*: If all systems support the same interface, customers can concentrate on the problem of choosing the implementation that best meets their own particular needs, without having to get involved in the additional complexity of choosing among different interfaces (possibly widely different interfaces).

On the other hand, there are some major disadvantages also:

- *A standard can stifle creativity*: Implementers may effectively be preempted from providing "the best" (or a good) solution to some problem, because the standard already prescribes some alternative, less satisfactory, solution to that same problem.
- *SQL in particular is very far from ideal as a relational language*: This criticism is elaborated in Appendix E. To quote from that appendix: " . . . it cannot be denied that SQL in its present form leaves rather a lot to be desired—even that, in some important respects, it fails to realize the full potential of the relational model." The basic problem (in this writer's opinion) is that, although there are well-established principles for the design of formal languages, there is little evidence that SQL was ever designed in accordance with any such principles. As a result, the language is filled with numerous restrictions, ad hoc constructs, and annoying special rules. These factors in turn make the language hard to define, describe, teach, learn, remember, apply, and implement.
- *Standard SQL especially is severely deficient in several respects*: In addition to the deficiencies mentioned under the previous point (i.e., deficiencies that are intrinsic to the original SQL language per se), standard SQL in particular suffers from certain additional deficiencies. Specifically, it fails to include any support at all for several functions that are clearly needed in practice (e.g., the DROP TABLE function), and it leaves as "implementation-defined" certain aspects that would be much better spelled out as part of the standard (e.g., the effect of certain operations on cursor position). As a result, it seems likely that every realistic implementation of the standard will necessarily include many implementation-defined extensions and variations, and hence that no two "standard" SQL implementations will ever be truly identical.

Despite these drawbacks, however, the fact is that the standard exists, vendors are scrambling to support it, and customers are demanding such support. Hence this book.

2

An Overview of SQL

2.1 INTRODUCTION

The aim of this chapter is to present a brief and very informal introduction to some of the major facilities of standard SQL, and thereby to pave the way for an understanding of the more formal and thorough treatment of the language in subsequent chapters. The chapter is loosely based on Chapter 1 ("Relational Database: An Overview") from the author's book *Relational Database: Selected Writings* (Addison-Wesley, 1986).

The function of the SQL language is to support the definition, manipulation, and control of data in a relational database. A *relational database* is simply a database that is perceived by the user as a collection of *tables*—where a table is *an unordered collection of rows* ("relation" is just a mathematical term for such a table). An example, the suppliers-and-parts database, is shown in Fig. 2.1. Tables S, P, and SP in that figure represent, respectively, suppliers, parts, and shipments of parts by suppliers. Note that each table can be thought of as a *file*, with the rows representing records and the columns fields. The SQL standard always uses the terms "row" and "column," however, never "record" and "field," and in this book we will therefore generally do likewise.

SQL "data manipulation" statements—i.e., SQL statements that perform data retrieval or updating functions—can be invoked either interactively or from within an application program. Figure 2.2 illustrates both

S	SNO	SNAME	STATUS	CITY
	S1	Smith	20	London
	S2	Jones	10	Paris
	S3	Blake	30	Paris
	S4	Clark	20	London
	S5	Adams	30	Athens

P	PNO	PNAME	COLOR	WEIGHT	CITY
	P1	Nut	Red	12	London
	P2	Bolt	Green	17	Paris
	P3	Screw	Blue	17	Rome
	P4	Screw	Red	14	London
	P5	Cam	Blue	12	Paris
	P6	Cog	Red	19	London

SP	SNO	PNO	QTY
	S1	P1	300
	S1	P2	200
	S1	P3	400
	S1	P4	200
	S1	P5	100
	S1	P6	100
	S2	P1	300
	S2	P2	400
	S3	P2	200
	S4	P2	200
	S4	P4	300
	S4	P5	400

Fig. 2.1 The suppliers-and-parts database (sample values)

(a) *Interactive invocation*:

```
SELECT S.CITY                Result:  CITY
FROM   S                              ------
WHERE  S.SNO = 'S4'                   London
```

(b) *Invocation from an application program (PL/I)*:

```
EXEC SQL SELECT S.CITY INTO :SC      Result:  SC
         FROM   S                             ------
         WHERE  S.SNO = 'S4' ;                London
```

Fig. 2.2 SQL retrieval example

cases; it shows a data retrieval operation (SELECT in SQL) being used both (a) interactively and (b) from within a PL/I program. In general, interactive invocation means that the statement in question is executed from an interactive terminal and (in the case of retrieval) the result is displayed at that terminal. Invocation from within an application program means that the statement is executed as part of the process of executing that program and (in the case of retrieval) the result is fetched into an input area within that program (":SC" in Fig. 2.2(b)). *Note*: The syntactic style illustrated in Fig. 2.2 for invoking SQL from an application program is not the only one possible. See Chapter 3.

One further remark: The reader will have noticed that we used qualified column names (S.SNO, S.CITY) in Fig. 2.2. SQL in fact allows qualifiers to be omitted in many contexts (including, in particular, the SELECT and