CHAPTER 10

Introduction to SQL

Abstract

This chapter presents an overview of the SQL database language. It covers the language's history, as well as the software environments that provide an interface to database management systems. It concludes with a discussion of the elements of a SQL statement.

Keywords

SQL
SQL history
SQL conformance levels
SQL command processors

SQL[1] is a database definition and database manipulation language that has been implemented by virtually every relational database management system (DBMS) intended for multiple users, partly because it has been accepted by ANSI (the American National Standards Institute) and ISO (International Standards Organization) as a standard query language for relational databases.

The chapter presents an overview of the environment in which SQL exists. We will begin with a bit of SQL history so you will know where it came from and where it is heading. Then, you will read about the way in which SQL commands are processed and the software environments in which they function.

A Bit of SQL History

SQL was developed by IBM at its San Jose Research Laboratory in the early 1970s. Presented at an ACM conference in 1974, the language was originally named SEQUEL (Structured English Query Language) and pronounced "sequel." The language's name was later shortened to SQL.

Although IBM authored SQL, the first SQL implementation was provided by Oracle Corporation (then called Relational Software Inc.). Early commercial implementations were concentrated on midsized UNIX-based DBMSs, such as Oracle, Ingres, and Informix. IBM followed in 1981 with SQL/DS, the forerunner to DB2, which debuted in 1983.

ANSI published the first SQL standard (SQL-86) in 1986. An international version of the standard issued by ISO appeared in 1987. A significant update to SQL-86 was released in 1989 (SQL-89). Virtually all relational DBMSs that you encounter today support most of the 1989 standard.

In 1992, the standard was revised again (SQL-92), adding more capabilities to the language. Because SQL-92 was a superset of SQL-89, older database application programs ran under the new standard with minimal modifications. In fact, until October 1996, DBMS vendors could submit their products to NIST (National Institute for Standards and Technology) for verification of SQL standard compliance. This testing and certification process provided significant motivation for DBMS vendors to adhere to the SQL standard. Although discontinuing standard compliance testing saves the vendors money, it also makes it easier for products to diverge from the standard.

The SQL-92 standard was superseded by SQL:1999, which was once again a superset of the preceding standard. The primary new features of SQL:1999 supported the object-relational data model, which is discussed in Chapter 27 of this book.

The SQL:1999 standard also adds extensions to SQL to allow scripts or program modules to be written in SQL or to be written in another programming language, such as C++ or Java and then invoked from within another SQL statement. As a result, SQL becomes less "relational," a trend decried by some relational purists.

Note: Regardless of where you come down on the relational theory argument, you will need to live with the fact that the major commercial DBMSs, such as Oracle and DB/2, have provided support for the object-relational data model for some time now. The object-relational data model is a fact of life, although there certainly is no rule that says that you must use those features, should you choose not to do so.

Even the full SQL:1999 standard does not turn SQL into a complete, stand-alone programming language. In particular, SQL lacks I/O statements. This makes perfect sense, since SQL should be implementation and operating system independent. However, the full SQL:1999 standard does include operations such as selection and iteration that make it *computationally complete*. These language features, which are more typical of general-purpose programming languages, are used when writing stored procedures and triggers:

- Triggers: A *trigger* is a script that runs when a specific database action occurs. For example, a trigger might be written to execute when data are inserted or deleted.
- Stored procedures: A *stored procedure* is a script that runs when it is called by an application program written in a general-purpose programming language or another SQL language module.

Triggers and stored procedures are stored in the database itself, rather than being a part of an application program. The scripts are, therefore, available to all application programs written to interact with the database.

The SQL standard has been updated four times since the appearance of SQL:1999, in versions named SQL:2003, SQL:2006, SQL:2008, and SQL:2011. As well as fleshing out the capabilities of the core relational capabilities and extending object-relational support, these revisions have added support for XML (Extensible Markup Language). XML is a platform-independent method for representing data using text files. SQL's XML features are introduced in Chapter 26.

XML features are introduced in Chapter 26.

Conformance Levels

The SQL in this book is based on the more recent versions of the SQL standard (SQL:2003 through SQL:2011). However, keep in mind that SQL:2011 (or whatever version of the language you are considering) is simply a standard, not a mandate. Various DBMSs exhibit different levels of conformance to the standard. In addition, the implementation of language features usually lags behind the standard. Therefore, although SQL:2011 may be the latest version of the standard, no DBMS meets the entire standard and most are based on earlier versions.

Note: In one sense, the SQL standard is a moving target. Just as DBMSs look like they're going to catch up to the most recent standard, the standard is updated. DBMS developers scurry to implement new features and, as soon as they get close, the standard changes again.

Conformance to early versions of the standard (SQL-92 and earlier) was measured by determining whether the portion of the language required for a specific level of conformance were supported. Each feature in the standard was identified by a *leveling rule*, indicating at which conformance level it was required. At the time, there were three conformance levels:

- Full SQL-92 conformance: all features in the SQL-92 standard are supported.
- Intermediate SQL-92 conformance: all features required for intermediate conformance are supported.
- Entry SQL-92: conformance: all features required for entry level conformance are supported.

In truth, most DBMSs were only entry level compliant, and some supported a few of the features at higher conformance levels. The later standards define conformance in a different way, however.

The standard itself is documented in nine parts (parts 1, 2, 3, 4, 9, 10, 11, 13, 14). Core conformance is defined as supporting the basic SQL features (Part 2, Core/Foundation) as well as features for definition and information schemas (Part 11, SQL/Schemata). A DBMS can claim conformance to any of the remaining parts individually, as long as the product meets the conformance rules presented in the standard.

In addition to language features specified in the standard, there are some features from earlier standards that, although not mentioned in the 2006, 2008, and 2011 standards, are widely implemented. This includes, for example, support for indexes.

SQL Environments

There are two general ways in which you can issue a SQL command to a database:

- Interactive SQL, in which a user types a single command and sends it immediately to the database: The result of an interactive query is a table in main memory (a *virtual table*). In mainframe environments, each user has one result table at a time, which is replaced each time a new query is executed; PC environments sometimes allow several. As you read in Chapter 6, result tables may not be legal relations—because of nulls, they may have no primary key—but that is not a problem because they are not part of the database, but exist only in main memory.
- Embedded SQL, in which SQL statements are placed in an application program: The interface presented to the user may be form-based or command-line based. Embedded SQL may be *static*, in which case the entire command is specified at the time the program is written. Alternatively, it may be *dynamic*, in which case the program builds the statement using user input and then submits it to the database.

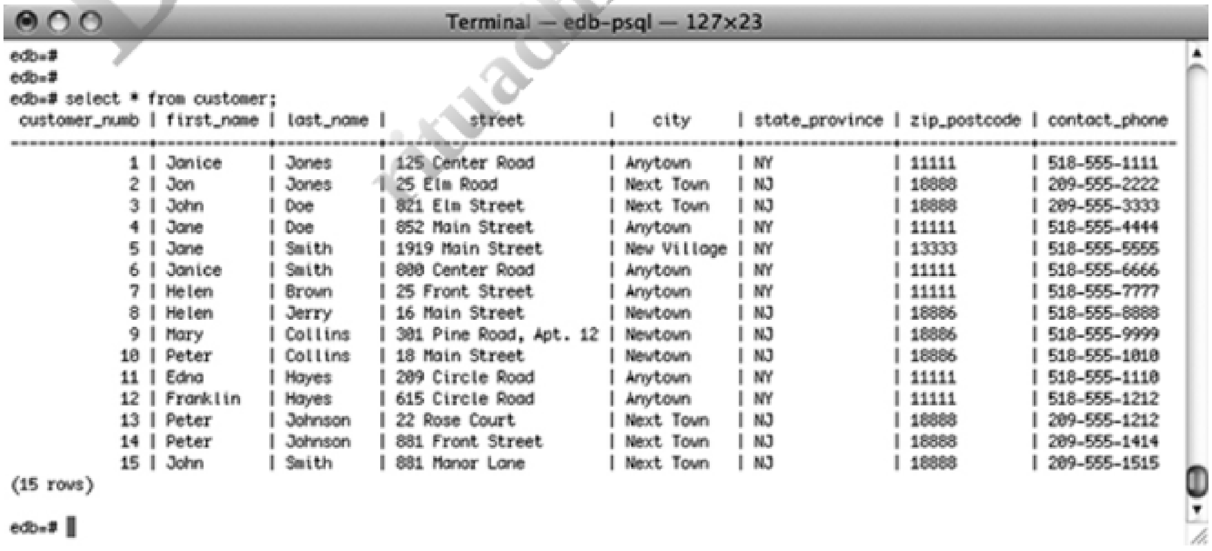
In addition to the two methods for writing SQL syntax, there are also a number of graphic query builders. These provide a way for a user who may not know the SQL language to "draw" the elements of a query. Some of these programs are report writers (for example, Crystal Reports[2]) and are not intended for data modification or for maintaining the structure of a database.

Interactive SQL Command Processors

At the most general level, we can describe working with an interactive SQL command processor in the following way:

- Type the SQL command.
- Send the command to the database and wait for the result.

In this era of the graphic user interface (GUI), command line environments like that in Figure 10.1 seem rather primitive. Nonetheless, the SQL command line continues to provide basic access to relational databases and is used extensively when developing a database.



```
edb=#
edb=#
edb=# select * from customer;
 customer_numb | first_name | last_name |        street          |    city     | state_province | zip_postcode | contact_phone
---------------+------------+-----------+------------------------+-------------+----------------+--------------+---------------
             1 | Janice     | Jones     | 125 Center Road        | Anytown     | NY             | 11111        | 518-555-1111
             2 | Jon        | Jones     | 25 Elm Road            | Next Town   | NJ             | 18888        | 209-555-2222
             3 | John       | Doe       | 821 Elm Street         | Next Town   | NJ             | 18888        | 209-555-3333
             4 | Jane       | Doe       | 852 Main Street        | Anytown     | NY             | 11111        | 518-555-4444
             5 | Jane       | Smith     | 1919 Main Street       | New Village | NY             | 13333        | 518-555-5555
             6 | Janice     | Smith     | 800 Center Road        | Anytown     | NY             | 11111        | 518-555-6666
             7 | Helen      | Brown     | 25 Front Street        | Anytown     | NY             | 11111        | 518-555-7777
             8 | Helen      | Jerry     | 16 Main Street         | Newtown     | NJ             | 18886        | 518-555-8888
             9 | Mary       | Collins   | 301 Pine Road, Apt. 12 | Newtown     | NJ             | 18886        | 518-555-9999
            10 | Peter      | Collins   | 18 Main Street         | Newtown     | NJ             | 18886        | 518-555-1010
            11 | Edna       | Hayes     | 209 Circle Road        | Anytown     | NY             | 11111        | 518-555-1110
            12 | Franklin   | Hayes     | 615 Circle Road        | Anytown     | NY             | 11111        | 518-555-1212
            13 | Peter      | Johnson   | 22 Rose Court          | Next Town   | NJ             | 18888        | 209-555-1212
            14 | Peter      | Johnson   | 881 Front Street       | Next Town   | NJ             | 18888        | 209-555-1414
            15 | John       | Smith     | 881 Manor Lane         | Next Town   | NJ             | 18888        | 209-555-1515
(15 rows)

edb=# ▌
```

FIGURE 10.1 A typical SQL command line environment.

A command line environment also provides support for ad hoc queries, queries that arise at the spur of the moment and are not likely to be

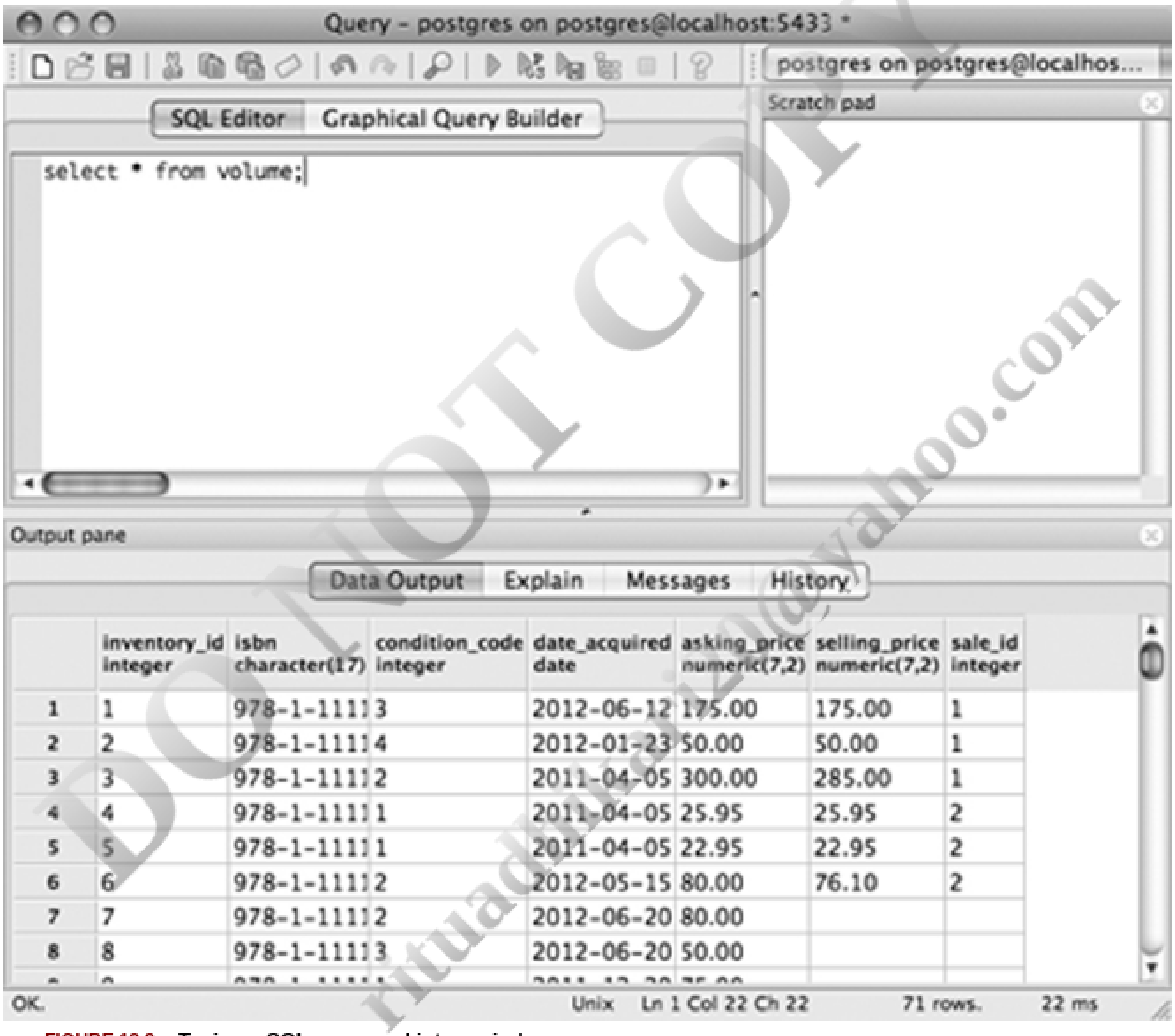
A command line environment also provides support for *ad hoc queries*, queries that arise at the spur of the moment, and are not likely to be issued with any frequency. Experienced SQL users can usually work faster at the command line than with any other type of SQL command processor.

The down side to the traditional command line environment is that it is relatively unforgiving. If you make a typing error or an error in the construction of a command, it may be difficult to get the processor to recall the command so that it can be edited and resubmitted to the database. In fact, you may have no other editing capabilities except the backspace key.

The SQL command examples that you will see throughout this book were all tested in a command line environment. As you are learning to create your own queries, this is, in most cases, the environment in which you will be working.

GUI Environments

There are actually two strategies used by GUI environments to provide access to a SQL database. The first is to simply provide a window into which you can type a command, just as you would do from the command line (for example, Figure 10.2, which shows a query tool for Postgres). Such environments usually make it easier to edit the command, supporting recall of the command and full-screen editing features.



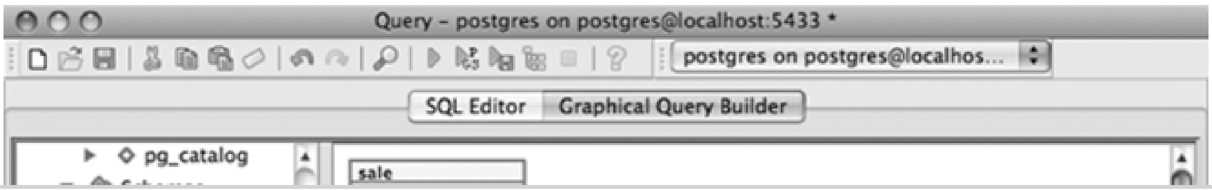
```
select * from volume;
```

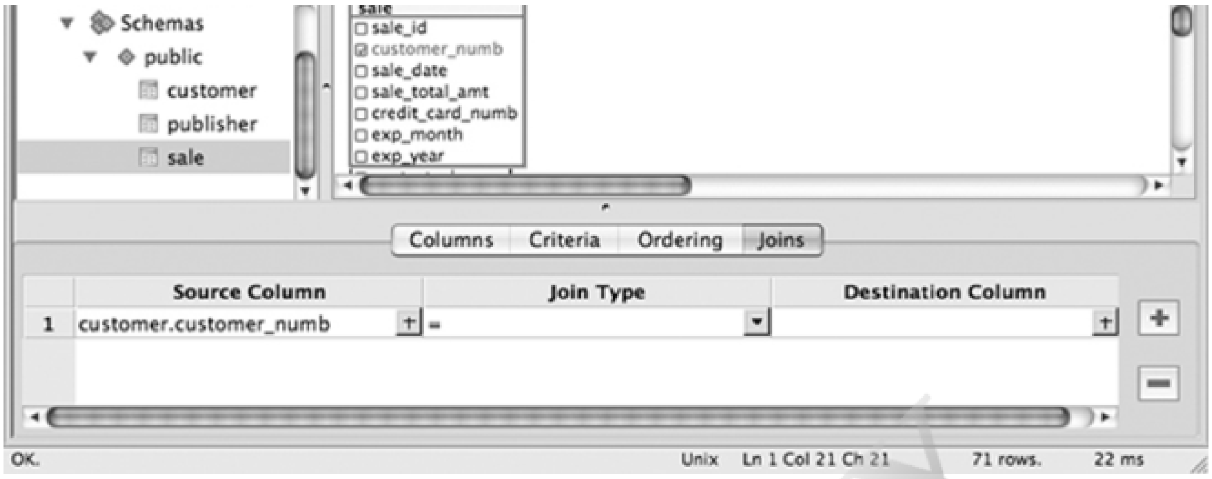
	inventory_id integer	isbn character(17)	condition_code integer	date_acquired date	asking_price numeric(7,2)	selling_price numeric(7,2)	sale_id integer
1	1	978-1-1111	3	2012-06-12	175.00	175.00	1
2	2	978-1-1111	4	2012-01-23	50.00	50.00	1
3	3	978-1-1111	2	2011-04-05	300.00	285.00	1
4	4	978-1-1111	1	2011-04-05	25.95	25.95	2
5	5	978-1-1111	1	2011-04-05	22.95	22.95	2
6	6	978-1-1111	2	2012-05-15	80.00	76.10	2
7	7	978-1-1111	2	2012-06-20	80.00		
8	8	978-1-1111	3	2012-06-20	50.00		

```
OK.                    Unix   Ln 1 Col 22 Ch 22        71 rows.      22 ms
```

FIGURE 10.2 **Typing a SQL command into a window.**

The other strategy is to provide a "query builder," an environment in which the user is guided through the construction of the query (for example, Figure 10.3). The query builder presents the user with lists of the legal command elements. Those lists change as the query is built so that the user also constructs legal syntax. The query builder type of SQL command environment makes it much easier for many users to construct correct SQL statements, but it is also slower than working directly at the command line.


Query – postgres on postgres@localhost:5433 *
postgres on postgres@localhos...
SQL Editor
Graphical Query Builder
pg_catalog
sale



	Source Column	Join Type	Destination Column
1	customer.customer_numb	=	

OK. Unix Ln 1 Col 21 Ch 21 71 rows. 22 ms

FIGURE 10.3 **A "query builder" environment.**

The Embedded SQL Dilemma

Embedding SQL in a general-purpose programming language presents an interesting challenge. The host languages (for example, Java, C++, JavaScript) have compilers or interpreters that don't recognize SQL statements as legal parts of the language. The solution is to provide SQL support through an application library that can be linked to a program. Program source code is passed through a precompiler that changes SQL commands into calls to library routines. The modified source code will then be acceptable to a host language compiler or interpreter.

In addition to the problem of actually compiling an embedded SQL program, there is a fundamental mismatch between SQL and a general-purpose programming language: Programming languages are designed to process data one row at a time, while SQL is designed to handle many rows at a time. SQL therefore includes some special elements so that it can process one row at a time when a query has returned multiple rows.

Elements of a SQL Statement

There are certainly many options for creating a SQL command. However, they are all made up of the same elements:

- Keywords: Each SQL command begins with a keyword—such as SELECT, INSERT, or UPDATE—that tells the command processor the type of operation that is to be performed. The remainder of the keywords precede the tables from which data are to be taken, indicate specific operations that are to be performed on the data, and so on.
- Tables: A SQL command includes the names of the tables on which the command is to operate.
- Columns: A SQL command includes the names of the columns that the command is to affect.
- Functions: A *function* is a small program that is built into the SQL language. Each function does one thing. For example, the AVG function computes the average of numeric data values. You will see a number of SQL functions discussed throughout this book.

Keywords and tables are required for all SQL commands. Columns may be optional, depending on the type of operation being performed. Functions are never required for a legal SQL statement, but, in some cases may be essential to obtaining a desired result.

For Further Reading

Chamberlin, DD, Raymond FB. SEQUEL: a structured english query language. 1974. http://www.almaden.ibm.com/cs/people/chamberlin/sequel-1974.pdf
Programmers Stack Exchange. Sequel vs. S-Q-L. http://programmers.stackexchange.com/questions/108518/sequel-vs-s-q-l
O'Reilly Media. History of the SQL Standard from SQL in a nutshell, 2nd ed. (entire book: http://users.atw.hu/sqlnut/index.html). http://users.atw.hu/sqlnut/sqlnut2-chp-1-sect-2.html

[1] Whether you say "sequel" or "S-Q-L" depends on how long you've been working with SQL. Those of us who have been working in this field for longer than we'd like to admit often say "sequel," which is what I do. When I started using SQL, there was no other pronunciation. That is why you'll see "a SQL" (a sequel) rather than "an SQL" (an es-que-el) throughout this book. Old habits die hard! However, many people do prefer the acronym.

[2] For more information, see www.crystalreports.com.