SQL

Don Chamberlin
IBM Almaden Research Center
San Jose, CA

SYNONYMS
SEQUEL, Structured Query Language

DEFINITION
SQL is the world's most widely-used database query language. It was developed at IBM Research Laboratories in the 1970's, based on the relational data model defined by E.F. Codd in 1970. It supports retrieval, manipulation, and administration of data stored in tabular form. It is the subject of an international standard named Database Language SQL.

HISTORICAL BACKGROUND

Early Language Development
In June 1970, E. F. Codd of IBM Research published a paper [1] defining the relational data model and introducing the concept of data independence. Codd's thesis was that queries should be expressed in terms of high-level, nonprocedural concepts that are independent of physical representation. Selection of an algorithm for processing a given query could then be done by an optimizing compiler, based on the access paths available and the statistics of the stored data; if these access paths or statistics should later change, the algorithm could be re-optimized without human intervention. In a series of papers, Codd proposed two high-level languages for querying relational databases, called relational algebra and relational calculus (also known as Data Sublanguage Alpha) [2].

The advantages of the relational model for application developers and database administrators were immediately clear. It was less clear whether an optimizing compiler could consistently translate nonprocedural queries into algorithms that were efficient enough for use in a production database environment. To investigate this issue, IBM convened a project called System R [3] at its research laboratory in San Jose, California. Between 1973 and 1979, this project designed and implemented a prototype relational database system based on Codd's ideas, testing and refining the prototype in several customer locations. SQL was the user interface defined by the System R research project. It later became the user interface for relational database products marketed by IBM and several other companies.

The principal goals that influenced the design of SQL were as follows:

1. SQL is a high-level, nonprocedural language intended for processing by an optimizing compiler. It is designed to be equivalent in expressive power to the relational query languages originally proposed by Codd.

2. SQL is intended to be accessible to users without formal training in mathematics or computer programming. It is designed to be typed on a keyboard. Therefore it is framed in familiar English keywords, and avoids specialized mathematical concepts or symbols.

3. SQL attempts to unify data query and update with database administration tasks such as creating and modifying tables and views, controlling access to data, and defining constraints to protect database integrity. In pre-relational database systems, these tasks were usually performed by specialized database administrators and required shutting down and reconfiguring the database. By building administrative functions into the query language, SQL helps to eliminate the database administrator as a choke point in application development.

4. SQL is designed for use in both decision support and online transaction processing environments. The former environment requires processing of complex queries, usually executed infrequently but accessing large amounts of data. The latter environment requires high-performance execution of parameterized transactions, repeated frequently but accessing (and often updating) small amounts of data. Both end-user interfaces and application programming interfaces are necessary to support this spectrum of usage.

The first specification of SQL was published in May 1974, in a 16-page conference paper [4] by Don Chamberlin and Ray Boyce, members of the System R project. In this paper, the language was named SEQUEL, an acronym for Structured English Query Language. The paper included a BNF syntax for the proposed language. This original paper presented only basic query features, without any facilities for data definition or update. However, the basic structure of the language, including query-blocks, grouping, set operations, and aggregating functions, has been consistent from this paper to the present day.

Over the course of the System R project, SEQUEL continued to evolve based on experience gathered by users and implementers. A much more complete description of the language was published in the IBM Journal of Research and Development in November 1976 [5], including data manipulation facilities (insert, delete, and update), a more complete join facility, facilities for defining tables and views, and database administration facilities including access control, assertions, and triggers. In 1977, because of a trademark issue, the name SEQUEL was shortened to SQL.

Although SQL was designed and prototyped at IBM Research, the language was published in the open literature, and the first commercial SQL product was released by a small company called Relational Software, Inc., in 1979. This product was named Oracle, a name that was later adopted by the company, which is no longer small. The first IBM product based on SQL was called SQL/Data System, released in 1981, followed by DB2, released in 1983 on mainframes and eventually supported on many IBM platforms. SQL has now been implemented by all major database vendors and is available in a wide variety of operating environments. In addition to commercial database products, SQL implementations include several popular open-source products such as MySQL [6] and Apache Derby [7].

Standards
Shortly after the first appearance of SQL in a commercial product, an effort was made to standardize the language. Over the years, the SQL standard has contributed to the growth of the database industry by defining a common interface for use by database vendors, application developers, and tools.

The first SQL standard, named "Database Language SQL," was published by the American National Standards Institute (ANSI) in 1986 (Standard No. X3.135-1986), and an identical standard with the same name was published by the International Standards Organization (ISO) in 1987 (Standard No. ISO 9075-1987). Over the years, ANSI and ISO have cooperated to keep their respective SQL standards synchronized as they have evolved through several versions.

The original standard, often called SQL-86, occupied just under 100 pages and included only simple queries, updates, and table definitions. It was followed in 1989 by a revised standard that added several kinds of constraints for protecting the integrity of stored data. This version of the standard comprised about 120 pages and is often referred to as SQL-89 ("Database Language SQL with Integrity Enhancement.")

A major revision of the SQL standard, usually called SQL-92, was published by ANSI and ISO/IEC in 1992. This version improved the orthogonality of the language, allowing expressions to be used wherever tables or scalar values are expected. SQL-92 also added several new features, including date and time datatypes, set-oriented operators such as UNION and INTERSECT, standard catalog tables for storing metadata, and schema-evolution features such as ALTER TABLE. SQL-92 comprised about 600 pages. A conformance test suite for SQL-92 was developed in the United States by the National Institute of Standards and Technology (NIST). After certifying several conforming products, NIST discontinued SQL conformance testing in 1996.

Between 1992 and 1999, two specialized extensions to SQL-92 were published, named Call Level Interface (CLI) and Persistent Stored Modules (PSM). CLI defines a set of functions whereby programs written in languages such as C can dynamically connect to relational databases and execute SQL statements. PSM extends SQL with assignment statements, control-flow statements, and exception handlers, making it possible to implement some database applications entirely in SQL. PSM was an attempt to standardize the procedural extensions such as PL/SQL [8] and Transact-SQL [9] that had been added to SQL by several database vendors.

The next major update of the SQL standard occurred in 1999 and is usually called SQL:1999. This new version split the standard into several parts, incorporating CLI as Part 3 and PSM as Part 4. SQL:1999 introduced important new functionality including triggers, large objects, recursive queries, and user-defined functions. It placed major emphasis on object-relational functionality and on new features for on-line analytic processing (OLAP). Details of these and other recent additions to SQL are described below under "Advanced Features." The sum of all the parts of SQL:1999 exceeded 2000 pages. Additional parts continue to be added to the SQL standard from time to time. The most recent major revision of the standard, known as SQL:2003, with all its parts, comprises more than 3600 pages.

Over the years, the SQL standard has provided a controlled framework within which the language can evolve to correct its initial limitations and to meet changing user requirements. The standard has also served to focus the industry's attention and resources, providing a common framework in which individuals and companies could develop tools, write books, teach courses, and provide consulting services. The standard has been only partially successful in making SQL applications portable across implementations; this goal has been hampered by the fact that different vendors have implemented different subsets of the standard, and by the lack (since 1996) of a test suite to validate conformance of an implementation. The latest versions of the various parts of the standard can be obtained from ISO [10] or from national standards organizations such as ANSI [11]. Jim Melton, editor of the SQL standard, has also published a two-volume reference book explaining the standard in a very accessible style [12].

SCIENTIFIC FUNDAMENTALS

Queries

SQL operates on data in the form of tables. Each table has a name and consists of one or more columns, each of which has a name and a datatype. The content of a table consists of zero or more rows, each of which has a value for each of the columns. The value associated with a given row and column may be an instance of the datatype of that column, or may be a special "null" value indicating that the value is missing (not available or not applicable). SQL statements, which may be queries or updates, operate on stored tables or on tabular "views" that are derived from stored tables. The result of a query is an unnamed virtual table. The result of an update is a change to the stored data, which is visible to subsequent statements. Generally, updates can be applied to a view only if each row in the view can be mapped uniquely onto a row of a stored table. This rule makes it possible to map updates on the view to updates on the underlying table.

An SQL query consists of one or more *query-blocks*. A query-block consists of several clauses, each of which begins with a keyword. Some of these clauses (keywords SELECT and FROM) are required, and others (keywords WHERE, GROUP BY, and HAVING) are optional. The examples in this article use upper-case keywords and lower-case names, although SQL is a case-insensitive language. The examples are based on two tables named PARTS and SUPPLIERS. The primary key of PARTS is PARTNO and the primary key of SUPPLIERS is SUPPNO. SUPPNO also appears as a foreign key in the PARTS table (see "Database Administration" below for definitions of primary key and foreign key.)

The following query-block illustrates a join of two tables. Conceptually, rows from the PARTS table are paired with rows from the SUPPLIERS table according to the criterion specified in the WHERE clause (SUPPNO's must match), and the resulting row-pairs are filtered by an additional condition (supplier's location must be Denver). From the surviving row-pairs, the SELECT clause specifies the columns that appear in the query result (in this case, the part number and the supplier name). Many different strategies are possible for executing this query; since SQL is a non-procedural language, choice of an execution strategy is left to an optimizing compiler.

```
SELECT p.partno, s.name
FROM parts p, suppliers s
WHERE p.suppno = s.suppno
AND s.location = 'Denver'
```

The following query-block illustrates grouping and aggregation. Conceptually, the rows of the PARTS table are partitioned into groups with matching SUPPNO values. The groups are then filtered by the HAVING clause, which applies a predicate based on group properties (in this case, retaining only groups that have at least ten rows.) Finally, the SELECT clause specifies the columns of the query result, which consists of one row for each group. In a grouping query, the selected expressions must consist of group properties (grouping keys or aggregating functions such as avg, sum, max, min, and count.) The SELECT clause can also specify names for the output columns. The query-block in this example returns, for each supplier that supplies at least ten parts, the average cost of all parts supplied by that supplier.

```
SELECT suppno, avg(cost) AS avgcost
FROM parts
GROUP BY suppno
HAVING count(*) >= 10
```

Many queries, including the above examples, consist of a single query-block. Query-blocks can also be combined to form larger queries. In the following example, a query-block serves as a subquery that computes a value used in another query-block. The subquery finds the maximum cost in the PARTS table, and the outer query-block returns the part numbers of the parts that have this maximum cost. The query-blocks are linked together by the IN keyword, denoting that the value in the WHERE-clause is tested for inclusion in the list of values returned by the subquery, which may in general return multiple values.

```
SELECT partno
FROM parts
WHERE cost IN
     (SELECT max(cost)
      FROM parts)
```

Another way in which query-blocks can be combined to form a larger query is by means of the set-operators UNION, INTERSECT, and EXCEPT, which compute the union, intersection, or difference of the sets of rows returned by two query-blocks (after eliminating duplicate rows). The datatypes of the rows returned by the respective query-blocks must be compatible. The following example computes the difference of two query-blocks to find the suppliers that supply no parts. It also includes an ORDER BY clause, which specifies an ordering for the rows in the query result. The ORDER BY clause applies to the whole query rather to an individual query-block.

```
   SELECT suppno
   FROM suppliers
EXCEPT
   SELECT suppno
   FROM parts
ORDER BY suppno
```

The first example above showed how the condition for joining two tables can be specified in a WHERE-clause. This method returns only data for which a matching row exists in both tables. Another join method called an *outer join* can be used to include rows from one of the joined tables that have no matching rows in the other table. The following example returns the supplier numbers of suppliers located in Denver, together with the parts that they supply, including Denver suppliers that supply no parts (the latter suppliers will appear in the query result with a null value in the PARTNO column):

```
SELECT s.suppno, p.partno
FROM suppliers s LEFT OUTER JOIN parts p
     ON s.partno = p.partno AND s.location = 'Denver'
```

Data Manipulation

The data manipulation facilities of SQL consist of the INSERT, DELETE, and UPDATE statements, which are used for inserting rows into a table, deleting rows from a table, and updating the values of rows in a table. The values to be used in data manipulation statements may be specified as constants or computed by subqueries, as illustrated by the following examples.

This example updates the PARTS table, increasing the cost of all the parts supplied by a certain supplier by ten percent:

```
UPDATE parts
SET cost = cost * 1.1
WHERE suppno = '105'
```

The following example deletes from the SUPPLIERS table all the suppliers that supply no parts. This example illustrates a subquery that is linked to the outer DELETE statement by the keywords NOT EXISTS. Notice that the subquery depends on a value in a row supplied by the outer UPDATE statement (referred to as s.suppno); therefore the subquery must be executed repeatedly for each row in the table being updated. This kind of subquery is called a *correlated subquery.*

```
DELETE FROM suppliers s
WHERE NOT EXISTS
     (SELECT partno
      FROM parts
      WHERE suppno = s.suppno)
```

Database Administration
In addition to queries and data manipulation, SQL provides facilities for performing database administration tasks. These tasks fall into three general categories: data definition, access control, and active data features.

SQL data definition facilities include statements for creating or dropping tables and views, and for adding or deleting columns of existing tables. The definition of a view takes the form of an SQL query specifying how the view can be derived from stored tables and/or other views.

SQL access control facilities are based on the concepts of users, privileges, and roles. A privilege is the ability to perform an action (such as select, insert, delete, or update) on an object (such as a table or, in some cases, a column of a table). A role is a set of privileges that can be granted to a set of users. In general, the user who creates an object can grant privileges on that object to individual users and to roles, and can also revoke these privileges. When granting a privilege, the grantor can specify whether the grantee is authorized to pass along the privilege to additional users or roles.

SQL active data facilities include constraints and triggers. Constraints serve to enforce database integrity by limiting the kinds of updates that can be applied to a table. Important kinds of constraints include the following:

- NOT NULL constraints, which prohibit null values in a specific column.
- CHECK constraints, which specify a predicate that must not be false for any row of the table.
- PRIMARY KEY constraints, which require that the values in a specific set of columns called the *primary key* uniquely identify a row of the table.
- FOREIGN KEY constraints, which require that, for each row of the constrained table, a specific set of columns called the *foreign key* contain only combinations of values that are also found in the primary key of a related table called the *parent table.* The definer of a FOREIGN KEY constraint can specify what happens when a data manipulation statement attempts to violate the constraint (for example, the violating statement might have no effect; or deletion of a row from the parent

table might cause the automatic deletion of related rows in the constrained table.) Enforcement of foreign key constraints is said to protect the *referential integrity* of stored data.

A trigger is an action that is automatically invoked whenever a specific event, called the triggering event, occurs. The definer of a trigger specifies the following properties:

- The triggering event, which may be insert, delete, or update of rows (or, in some cases, columns) of a specific table.
- Whether the trigger is invoked before or after the triggering event becomes effective.
- If the triggering event affects multiple rows, whether the trigger is invoked once for each affected row or only once for the whole triggering event.
- An optional trigger condition: a predicate that must be true at the time of the triggering event in order for the trigger to be activated.
- The trigger body: one or more SQL statements that are automatically executed when the triggering event occurs and the trigger condition is true. The trigger condition and the trigger body have access to special variables that contain the data values before and after the triggering event.

Constraints and triggers are useful for specifying and enforcing the semantics of stored data. In general, it is preferable to specify a given semantic rule by means of a constraint rather than a trigger if possible, since constraints apply to all kinds of actions and provide maximum opportunities for optimization. On the other hand, some kinds of semantic rules (for example, "salaries never decrease") can only be specified by using triggers.

Advanced Features

Over the years, a great deal of functionality has been added to the SQL language. The full set of SQL features is far too large and complex to be explained here. The following are some of the major areas in which advanced functionality has been added to SQL:

- **Recursion:** A recursive query consists of an initial subquery that computes some preliminary results and a recursive subquery that computes additional results based on values that were previously computed. The recursive subquery is executed repeatedly until no additional results are computed. Recursion is useful in queries that search some space for an optimum result, such as "Find the cheapest combination of flight segments to travel from Shanghai to Copenhagen." Recursive queries were first defined in SQL:1999.
- **OLAP:** Online analytic processing (OLAP) is used by businesses to analyze large volumes of data to identify facts and trends that may affect business decisions. The GROUP BY clause and aggregating functions (sum, avg, etc.) of early SQL provided a primitive form of OLAP functionality, which was greatly extended in later versions of the language. For example, the ROLLUP facility enables a query to apply aggregating functions at multiple levels (such as city, county, and state). The CUBE facility enables data to be aggregated along multiple dimensions (such as date, location, and category) within a single query. The WINDOW facility allows aggregating functions to be applied to a "moving window" as it passes over a collection of data. These facilities, and others, were introduced by SQL:1999 and enhanced in subsequent versions of the standard.
- **Functions and procedures:** Originally, SQL supported a fixed collection of functions, which grew slowly over the years. SQL:1999 introduced a capability for users to define additional functions and procedures that can be invoked from SQL statements. (In this context, a procedure is simply a function that is invoked by a CALL statement and that is not required to return a value.) The bodies of user-defined functions and procedures can be written either in SQL itself or in a host language such as C or Java.

- **Object-relational features:** Early versions of SQL could process data conforming to a fixed set of simple datatypes such as integers and strings. Over the years, a few additional datatypes such as dates and "large objects" were added. In the late 1990's, requirements arose for a more extensible type system. SQL:1999 introduced facilities for user-defined structured types and methods. These facilities support limited forms of object-oriented functionality, including inheritance and polymorphism.
- **Multi-media:** In 2000, the SQL Standard was augmented by a separate but closely-related standard called "SQL Multimedia and Application Packages" (ISO/IEC 13249:2000), often referred to as SQL/MM. This new standard used the object-relational features introduced by SQL:1999 to define specialized datatypes and methods for text, images, and spatial data.
- **XML-related features:** XML is an increasingly popular format for data exchange because it mixes metadata (tags) with data, making the data self-describing. The popularity of XML has led to requirements to store XML data in relational databases and to convert data between relational and XML formats. These requirements have been addressed by a facility called SQL/XML, which was introduced as Part 14 of SQL:2003 and was updated in 2006. SQL/XML includes a new XML datatype, a set of functions for converting query results into XML format, and a feature whereby SQL can invoke XQuery as a sublanguage for processing stored XML data.

Criticisms

Like most widely-used programming interfaces, SQL has attracted its share of criticism. Issues that have been raised about the design of SQL include the following:

- The earliest versions of SQL lacked support for some important aspects of Codd's relational data model such as primary keys and referential integrity. These concepts were added to the language in SQL-89, along with other integrity-related features such as unique constraints and check-constraints.
- The earliest versions of SQL had some ad-hoc rules about how various language features could be combined, and lacked the closure property because the columns of query results did not always have names. These problems were largely corrected by SQL-92.
- Null values are a complex and controversial subject. One of Codd's famous "twelve rules" requires relational database systems to support a null value, defined as a representation of missing or inapplicable information that is systematic and distinct from all regular values [13]. Some writers believe that the complexity introduced by null values outweighs their benefit. However, there seems to be no method for dealing with missing data that is free of disadvantages. The SQL approach to this issue has been to support a null value and to allow database designers to specify, on a column-by-column basis, where nulls are permitted. One benefit of this approach has been that null values have proven useful in the design of various language features, such as outer join, CUBE, and ROLLUP, that have been added during the evolution of SQL.
- Unlike Codd's definition of the relational data model, SQL permits duplicate rows to exist, either in a database table or in the result of a query. SQL also allows users to selectively prohibit duplicate rows in a table or in a query result. The intent of this approach is to give users control over the potentially expensive process of duplicate elimination. In some applications, duplicate rows may be meaningful (for example, in a point-of-sale system, a customer may purchase several identical items in the same transaction.) As in the case of nulls, the SQL approach has been to provide users with tools to allow or disallow duplicate rows according to the needs of specific applications.
- Another source of criticism has been the "impedance mismatch" between SQL and the host languages such as C and Java in which it is often embedded. Exchanging data between two languages with different type systems makes applications more complex and interferes with global

optimization. One approach to this problem has been the development of computationally complete SQL-based scripting languages such as PSM.

KEY APPLICATIONS

SQL is designed to be used in a variety of application environments.

Most SQL implementations support an interactive interface whereby users can compose and execute ad-hoc SQL statements. In many cases, a graphical user interface is provided to display menus of available tables and columns and help the user to construct valid statements. These systems also usually support menu-based interfaces for administrative functions such as creating and dropping tables and views.

More complex applications usually involve use of both SQL and a host programming language. This requires a mapping between the type systems of SQL and the host language, and a well-defined interface for exchanging data between the two environments. Interfaces have been defined between SQL and C, Java, and many other host languages. These interfaces fall into two major categories:

- **Embedded SQL:** In this approach, SQL statements are embedded syntactically in the host program, and are processed by a precompiler that extracts the SQL statements and converts them to an optimized execution plan that is invoked when the host program is executed.
- **Call interfaces:** In this approach, the host program uses function calls to establish a connection to a database and to pass SQL statements to the database system for execution. Conversion of the SQL statements to optimized execution plans is done at runtime. The most widely-used interfaces of this type are ODBC (Open Database Connectivity) [14] and JDBC (Java Database Connectivity) [15]. Since ODBC and JDBC drivers exist for many popular relational database systems, these interfaces are widely used by applications that need to access remote databases or need to be compatible with database systems from multiple vendors.

Modern web-based database applications often employ a three-tiered architecture. The *client tier* handles user interaction and communicates with a remote server via a protocol such as HTTP. Application logic resides on this server, called the *mid-tier*. The application logic, in turn, accesses an SQL database using ODBC or JDBC. The database may be implemented on the mid-tier or on a separate machine called the *database tier* or "back-end".

CROSS REFERENCES

Database Trigger, Expressiveness of Query Languages, Integrity Constraints, Java Database Connectivity, Join, Key, Metadata, Null Values, Online Analytical Processing, Open Database Connectivity, Query Language, Query Optimization, Query Processing, Relational Algebra, Relational Calculus, Relational Model, Relational Query Language, SQL Isolation Levels, SQUARE, Transaction, Two-tier and Three-tier Architectures, Views, XQuery.

RECOMMENDED READING

[1] E. F. Codd. "A Relational Model of Data for Large Shared Databanks." *Communications of the ACM,* 13 (June 1970) p. 377.

[2] E. F. Codd. "A Data Base Sublanguage founded on the Relational Calculus." *Proc. ACM SIGFIDET Workshop,* San Diego, CA, Nov. 1971.

[3] M. Astrahan et al. "System R: A Relational Approach to Database Management." *ACM Transactions on Database Systems,* 1 (June 1976) p. 97.

[4] D. Chamberlin and R. Boyce. "SEQUEL: A Structured English Query Language." *Proc. ACM SIGFIDET Workshop,* Ann Arbor, MI, May 1974. See also http://www.almaden.ibm.com/cs/people/chamberlin/sequel-1974.pdf.

[5] D. Chamberlin et al. "SEQUEL 2: A Unified Approach to Data Definition, Data Manipulation, and Control." *IBM Journal of Research and Development,* 20 (Nov. 1976), p. 560.

[6] MySQL. See http://www.mysql.com.

[7] Apache Derby. See http://db.apache.org/derby/.

[8] S. Feuerstein and B. Pribyl. *Oracle PL/SQL Programming, 4th Edition.* O'Reilly, 2005.

[9] *Transact-SQL Reference,* in Microsoft SQL Server Development Center. See http://msdn2.microsoft.com/en-us/library/ms189826.aspx.

[10] *Database Language SQL.* Standard ISO/IEC 9075-1, -2, etc. Contact ISO at http://www.iso.ch.

[11] *Database Language SQL.* Standard ANSI/ISO/IEC 9075-1, -2, etc. Contact ANSI at http://www.ansi.org.

[12] J. Melton and A. R. Simon. *SQL:1999–Understanding Relational Language Components.* Morgan Kaufmann, 2002. Also see J. Melton, *Advanced SQL:1999–Understanding Object-Relational and Other Advanced Features.* Morgan Kaufmann, 2003.

[13] E. F. Codd. "Does Your DBMS Run by the Rules?" *ComputerWorld,* 21, Oct. 1985. See also http://en.wikipedia.org/wiki/Codd's_12_rules.

[14] R. E. Sanders. *ODBC 3.5 Developer's Guide.* McGraw-Hill, 1998.

[15] *JDBC Overview,* in Sun Developer Network. See http://java.sun.com/products/jdbc/overview.html.