**What is SQL? Structured Query Language explained**

SQL is neither the fastest nor the most elegant way to talk to databases, but it is the best way we have; here’s why

Today, Structured Query Language is the standard means of manipulating and querying data in relational databases, though with proprietary extensions among the products. The ease and ubiquity of SQL have even led the creators of many “[NoSQL](https://www.infoworld.com/article/2617405/nosql/7-hard-truths-about-the-nosql-revolution.html)” or non-relational data stores, such as [Hadoop](https://www.infoworld.com/resources/16257/hadoop/download-the-hadoop-deep-dive), to adopt subsets of SQL or come up with their own SQL-like query languages.

But SQL wasn’t always the “universal” language for relational databases. From the beginning (circa 1980), SQL had certain strikes against it. Many researchers and developers at the time, including me, thought that the overhead of SQL would keep it from ever being practical in a production database.

Clearly, we were wrong. But many still believe that, for all of SQL’s ease and accessibility, the price exacted in runtime performance is often too high.

**Before SQL**

Before there was SQL, databases had tight, navigational programming interfaces, and typically were designed around a network schema called the [Codasyl](https://en.wikipedia.org/wiki/CODASYL) data model. Codasyl (Committee on Data Systems Languages) was a consortium that was responsible for the Cobol programming language (starting in 1959) and database language extensions (starting 10 years later).

When you programmed against a Codasyl database, you were navigating to records through sets, which express one-to-many relationships. Older hierarchical databases only allow a record to belong to one set. Network databases allow a record to belong to multiple sets.

Say you wanted to list the students enrolled in CS 101. First you would find "CS 101" in the Courses set by name, set that as the owner or parent of the Enrollees set, find the first member (ffm) of the Enrollees set, which is a Student record, and list it. Then you would go into a loop: Find next member (fnm) and list it. When fnm failed, you would exit the loop.

That may seem like a lot of scut work for the database programmer, but it was very efficient at execution time. Experts like Michael Stonebraker of University of California at Berkeley and Ingres pointed out that doing that sort of query in a Codasyl database such as [IDMS](https://en.wikipedia.org/wiki/IDMS) took roughly half the CPU time and less than half the memory as the same query on a relational database using SQL.

For comparison, the equivalent SQL query to return all of the students in CS 101 would be something like

SELECT student.name FROM courses, enrollees, students WHERE course.name ="CS 101"

That syntax implies a relational inner join (actually two of them), as I’ll explain below, and leaves out some important details, such as the fields used for the joins.

**Relational databases and SQL**

Why would you give up a doubling of execution speed and memory? There were two big reasons: ease of development and portability. I didn’t think either one mattered much in 1980 compared to performance and memory requirements, but as computer hardware improved and became cheaper people stopped caring about execution speed and memory and worried more about the cost of development.

In other words, Moore’s Law killed Codasyl databases in favor of relational databases. As it happened, the improvement in development time was significant, but SQL portability turned out to be a pipe dream.

Where did the relational model and SQL come from? [E.F. “Ted” Codd](https://en.wikipedia.org/wiki/Edgar_F._Codd) was a computer scientist at the IBM San Jose Research Laboratory who worked out the theory of the relational model in the 1960s and published it in 1970. IBM was slow to implement a relational database in an effort to protect the revenues of its Codasyl database IMS/DB. When IBM finally started its System R project, the development team (Don Chamberlin and Ray Boyce) wasn’t under Codd, and they ignored Codd’s 1971 Alpha relational language paper to design their own language, Sequel (Structured English Query Language). In 1979, before IBM had even released its product, Larry Ellison incorporated the language in his Oracle Database (using IBM’s pre-launch Sequel publications as his spec). Sequel soon became SQL to avoid an international trademark violation.

The “tom-toms beating for SQL” (as Michael Stonebraker put it) were coming not only from Oracle and IBM, but also from customers. It wasn’t easy to hire or train Codasyl database designers and programmers, so Sequel (and SQL) looked much more attractive. SQL was so attractive in the later 1980s that many database vendors essentially stapled a SQL query processor on top of their Codasyl databases, to the great dismay of Codd, who felt that relational databases had to be designed from scratch to be relational.

A pure relational database, as designed by Codd, is built on tuples grouped into relations, consistent with first-order predicate logic. Real-world relational databases have tables that contain fields, constraints, and triggers, and tables are related through foreign keys. SQL is used to declare the data to be returned, and a SQL query processor and query optimizer turn the SQL declaration into a query plan that is executed by the database engine.

SQL includes a sub-language for defining schemas, the data definition language (DDL), along with a sub-language for modifying data, the data manipulation language (DML). Both of these have roots in early Codasyl specifications. The third sub-language in SQL declares queries, through the SELECT statement and relational joins.